Performance Measurement Tools for SIP Server

Samit Jain
Columbia University, New York
sj2195@cs.columbia.edu

TABLE OF CONTENTS

1. ABSTRACT 3
2. INTRODUCTION 4
3. PERFORMANCE ISSUES 6
4. ARCHITECTURE 10
5. MEASUREMENTS AND RESULTS 12
6. FUTURE WORK 19
7. REFERENCES 20
A. PROGRAM DOCUMENTATION 21
B. EXAMPLES 22

Abstract

Columbia **I**nterNet **E**xtensible **M**ultimedia **A**rchitecture (CINEMA) is a set of SIP-based Internet multimedia servers for creating Internet telephony and multimedia system. It consists of a number of components such as SIP proxy, redirect and registrar server, SIP presence server, SIP multimedia conferencing server. This system needs to be appropriately dimensioned and provisioned for different types of workloads and requirements. SIPstone defines a basic set of metrics for evaluating and benchmarking SIP proxy, redirect and registrar servers and attempts to measure the request handling capacity of a SIP server or a cluster of SIP servers.

The main goal of this project is to extend the SIPstone set of tools for testing SIP server with different transport protocols – TCP, UDP, and TLS, with or without authentication, and SIP presence server performance measurement. Presence server testing is defined in SIMPLEstone and includes SUBSCRIBE-NOTIFY and PUBLISH-NOTIFY tests. The report is organized into following sections:

1) Introduction
2) Performance Issues
3) Architecture
4) Types of Tests
5) Program Documentation
6) Results
7) Future Work
8) References

Introduction

Before we look at the details of the benchmarking and testing mechanisms, it is necessary to review the SIP servers and their operations.

The SIP server (`sipd`) [3] or server under test (SUT) as referred by in this report, is a redirect, forking[1] proxy, and registration server that provides name mapping, user location and scripting services[2]. It also allows users to register their current location with the server. The SIP location or registration servers maintain the locations where users could be reached.

The server currently understands the ACK, BYE, CANCEL, INVITE, OPTIONS and REGISTER requests. Invitations and registrations are authenticated using digest authentication. The details of individual SIP methods can be found in [4]. The list of audio/video algorithms supported and the transport addresses to receive them, are described using Session Description Protocol [5], carried as the body of SIP requests and responses.

Presence server (`pa`) [6] is a separate server that allows users to subscribe to each other's presence and receive presence notification. The users (watchers) subscribe to presence information of others (presentities) using SIP SUBSCRIBE and receive notification about changes in state via SIP NOTIFY messages. Presence information can be published to the presence server using SIP PUBLISH message. The subscriber can specify *watcher filters* to enable filtering the events for which it wants to receive notifications or filtering the content of notifications. The notification is delivered to the watcher in the form of a flexible XML-based *presence document* [6], which is created by the presence server depending on the composition policy and privacy filters. Subscriptions, publications and notifications are authenticated using standard SIP authentication mechanisms [4].

The main objective of the benchmark suite is to measure the request handling capacity of the server for different types of SIP requests and configuration settings. The current implementation performs tests using a pre-configured workload as specified in the configuration file, which is flexible enough to allow us to exercise a breadth of system components, such as concurrent initiation of calls, call intervals, transport protocol, connection types, authentication, etc. Different types of requests and settings will differ in the impact they have on the server processing capacity and the network bandwidth, ultimately affecting the request handling capacity of the server. For example, a PUBLISH request entails a much higher load on the server and network as compared to an INVITE request, since the former depends not only on the user population and request rate, but also on the number of subscribers, composition policy, privacy filters, and subscriber filters[3].

[1]Forking is currently not being tested since it is hard to predict behavior of common forking proxies.
[2]Programmable scripts are currently not being tested since services vary for different applications.
[3]Performance testing of different composition policies is not implemented.

The benchmarking tool has some limitations. The results obtained can be severely dependent on the specific characteristics of the test-bed and test-environment. It may not be possible to derive results of general validity, but our main goal is to estimate the relative performance of different test configurations.

Some of the objectives of the benchmark suite are:

1. Characterizing server performance for dimensioning and provisioning
2. The system should allow maximum configuration with simplicity
3. Repeatability of test results

Note that the benchmark suite cannot be used to test server functionality, or evaluate protocol compliance and robustness.

Some of the results obtained from our tests with different configurations can be summarized in the following table.

	REGISTER	INVITE
UDP	2000	1400
TCP	1200	800
TLS	720	400

Table 1: Performance in terms of request handling capacity (req/sec)

There has been considerable work being done currently and which has been done in the past on SIPstone and SIP performance measurements. [1] describes the original SIPstone specification. [2] is ongoing work on performance testing the presence server. [8] describes failover and load sharing in SIP telephony.

In the next section, we discuss some of the performance issues related to SIP servers.

Performance Issues

1. User Population

The performance of a SIP proxy and registration server is directly dependent on the user population. Most SIP methods such as for data update (REGISTER) and for data lookup (INVITE) require database interactions and authentication lookups. There might also be other database transactions for logging, etc., within the same SIP transaction processing. This performance is likely to be dependent more on the back-end database server than on the SIP server. The number of users can be specified in the configuration file and it is likely that the user population is large, close to thousands of users.

This factor depends highly on the architecture of specific database system being deployed. The current benchmark uses a MySQL backend server. Also different database architectures have other implementation tradeoffs, which might affect its performance and reliability under varying conditions.

The performance of presence server depends more on other factors such as request rate, composition policies and size of filters, than on the user population, since it uses an in-memory hash table to store users. But it is likely that the impact of user population on performance of presence server will only increase with more users. Each call to presence server also requires authentication lookup, storing or retrieving subscriptions, contacts lookup, and other auxiliary transactions. Also the number of subscribers per presentity has an impact on server performance.

In order to see the impact of user population on performance of SIP servers, a separate test database should be created for new test run. Presently, we use a central database system for all the tests. This prevents us from demonstrating the affect of user population on performance for different number of users specified in the test configuration, e.g., 10,000 vs. 100. As part of the future work, a separate test database instance should be created for each new test run.

2. Request Rate

Request rate is defined as the number of requests sent to the server per second. The performance of the SIP proxy server is directly related to the request rate. Each SIP server has a threshold capacity beyond which the performance goes below optimal or below an acceptable limit, which can be low success rate or high turn around time. This can be configured in the test configuration file. We need to measure the transaction duration for a request and ignore the call duration since SIP proxy servers only maintain transaction state and not call state.

The request rate has significant impact on the performance of presence server. Since processing involved for each request type PUBLISH or SUBSCRIBE is different; the performance depends on the relative rate of each type of request, specifically, subscription refresh rate and publication rate.

3. Request Type

The type of request affects the performance on SIP server. Each request type causes different processing on the server and entails different load on the server and on the network. For example, a REGISTER request causes the SIP server to a) authorize the request, b) store the association in the database, and c) respond with 200 OK response. On the other hand, for a PUBLISH request, the presence server needs to a) authorize the request, b) retrieve subscribers from database, c) retrieve composition policy for the presentity and filters for each subscriber, d) compose a presence document, e) for each subscriber, apply privacy filters, f) for each subscriber, apply watcher filters, g) send 200 OK response to publisher, h) send NOTIFY to each subscriber. This clearly shows the difference between the processing of REGISTER and PUBLISH request types. The INVITE request has a different requirement as it causes the SIP proxy server to maintain transaction state until it gets response from the callee.

In the benchmark, we test registrar, proxy and presence server separately, but it would be interesting to test with a mixed load of these request types and measure the affect of varying request rates between different request types in the same test. We could model different systems such as for call center, mobile users, with varying invitation and registration rates. This has been proposed in SIPstone draft [1] but not implemented. This is useful to test the usage of a particular server implementation for a particular system and helps provisioning system and network resources.

For presence server, the request handling capacity of the server depends on the subscription refresh rate and publication rate. The latter has a bigger impact since PUBLISH imposes more overhead on server and network. We measure the request handling capacity for each of the message types in terms of successful SUBSCRIBE-NOTIFY and PUBLISH-NOTIFY calls.

4. Transport Protocol

SIP operates independently of the underlying transport protocol, which might be TCP, UDP, TLS, etc. However performance varies with different protocols. There is usually a tradeoff between performance vs. reliability and security while selecting a transport protocol. Ideally, we first select a transport protocol to use depending on system requirements and then perform optimizations for that protocol. When packet loss is low and the size of SIP messages is within maximum transmission unit (MTU), SIP over UDP is most efficient since it does not require any flow control or connection. Otherwise, we need to use TCP to guarantee message transmission reliability. When using TCP, issues such as the ability of the server to handle a large number of open connections and its use of persistent connections need to be considered. If we need to ensure confidentiality, integrity and privacy, we need to use SIP over TLS. When using TLS, we need to consider capability of server to handle large number of SSL sessions and TCP connections. We can specify the transport protocol in the configuration file. The performance measurement for different transport protocols and their comparison is shown in section 5.

5. Connection Policy

When using UDP, we do not open connections on the server. Network packet losses do not affect the server load directly, but cause retransmissions of requests, which can increase server processing load and amplify network traffic. If the packets losses are below 5%, we can use UDP to achieve good voice quality and acceptable call setup delays. If the packet losses are much higher, then we should probably be using TCP.

When using TCP, issues such as the ability of the server to handle a large number of *open* connections and its use of *persistent* connections need to be considered. We currently do not have a good estimate as to the typical number of upstream servers reaching a high-volume proxy server. If the number is low, almost all such servers will use persistent connections, otherwise we might have to use a separate connection[1] for each new request. When using persistent connections, we still get two connections between SIP proxies for requests sent in each direction, since source port is ephemeral and connections accepted at the transport layer cannot be reused for reverse connections. If this leads to potential scaling and performance problems, we could *reuse connections* opened in either direction as proposed in [7][2].

When we need to use TLS, we need to consider the ability of the server to handle SSL sessions. Since SSL session setup is expensive due to imposed network overhead and cryptographic operations required for a full TLS handshake, it is necessary to cache SSL sessions between SIP proxy servers. Depending on the load on the server, we might not be able to cache all SSL sessions. In practice, it might be beneficial to cache sessions between two proxy servers with high communication volume between them, than caching sessions between, say, a UAC and proxy server. Session timeout for SSL sessions needs to be high enough to span many calls or transactions.

6. Authentication

SIP authentication is based on stateless challenge-based authentication scheme similar to HTTP. The benchmark implements digest authentication scheme, which is most commonly used. Since the authentication scheme is stateless, they do not entail much burden on the server. However, if the user agents do not cache authentication credentials across multiple requests or dialogs, it leads to increased network traffic and more number of requests hitting the server. Therefore, it is highly recommended that user agents cache credentials associated with a specific realm. Authentication can be enabled or disabled in configuration file. We can also compare measurement results when credentials are cached vs. not cached by setting `preloadCredentials` parameter in the configuration file. Authentication credentials when not cached doubles the amount of network traffic.

[1]The connections should still be atleast kept open for an implementation-defined time to make it likely that transactions are completed over the same connection on which they are initiated, e.g. INVITE to ACK.
[2]Not currently implemented.

7. Composition and Filtering (only for presence server)

The composition policies on the server have a significant impact on the performance of the presence server. Firstly, the application of the policy, that is, whether different policies are applied to presentities or there is a single policy in place, affects the request handling capacity of the server. There is little gain in introducing per watcher composition policy, as noted in [6], and it introduces additional complexity and burden for server. Different types of composition policies and operations such as a simple union with replacement, or based on a more complex composition policy language, will impose different processing overheads on the server. The privacy and watcher filtering feature, the size of filter documents, which in turn determine the look up, comparison and xml manipulation operation on the server affect the server's performance. It also affects the amount of traffic generated by the server. The presence document format (rpid or pidf) also affects the amount of processing on server.

Architecture

The "server under test" (SUT) is a SIP proxy, redirect, registrar, or presence server whose performance is to be estimated. The benchmark consists of a set of SIPstone load generators that create the SIP request load, a call handler that simulates a user agent server and a central benchmark manager ("controller") that coordinates the execution of the benchmark, and the SUT. The call handlers may run along with the load generators or on different systems. Benchmarking consists of a series of test runs with increasing load levels generated by the load generators, saturating the processing capability of the server, and measuring its maximum throughput.

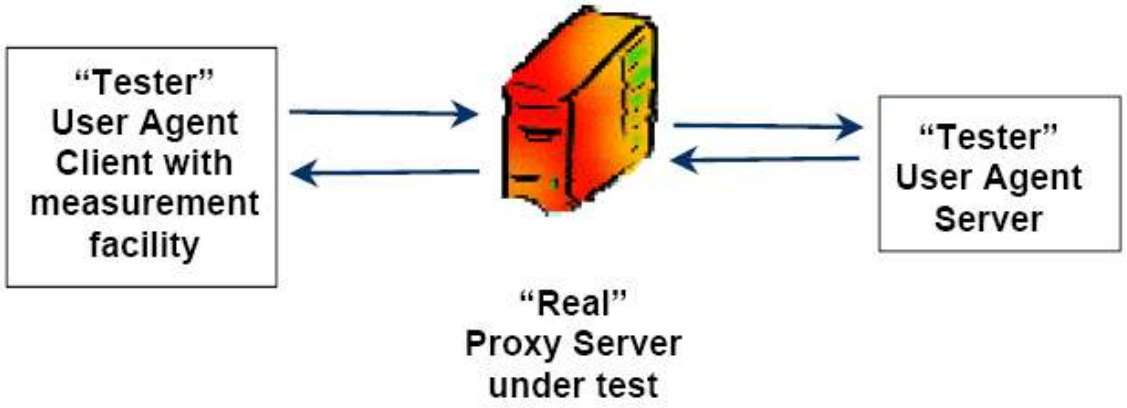


Figure 1: Basic Architecture for SIP server testing

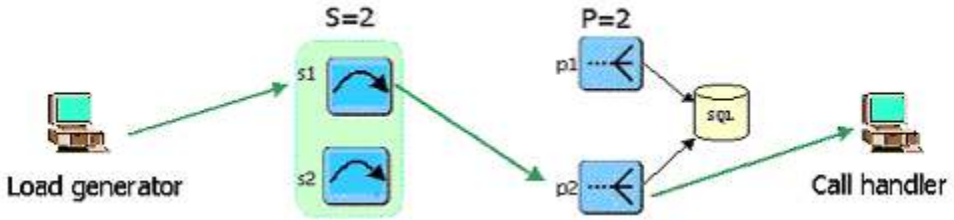


Figure 2: Testing in 2-stage clustering with 2 stateless, 2 stateful servers

Figure 1 shows the general architecture for single server testing. The SUT includes all machines and network connections spanning SIP server components including registrar, database, proxy servers, presence servers, etc. All machines including loaders, callhandlers, and SUT are connected on a 100 Mb/s Ethernet that is otherwise lightly loaded. All the tests were performed on IRT cluster machines to ensure minimal non-test traffic.

Figure 2 shows a sample configuration for 2-stage clustering. Presently, clustered testing is limited to a two-stage scaling architecture. The first set of proxy servers perform stateless request routing to a particular second-stage cluster server. The second-stage server performs the actual request processing. This architecture can scale to any desired processing load and user population size [8]. The stateless servers forward the requests to stateful servers based on user identifier, so that the requests are uniformly distributed amongst the stateful servers.

Please see Appendix A for information on running the tests in single-server mode or in a clustered setting.

Types of tests

1. **REGISTER**
    1. The loader sends REGISTER request to the server.
    2. Server stores registration info and responds with "200 OK" response.

2. **INVITE**
    1. Callhandler registers test users.
    2. Loader sends INVITE request.
    3. Server forwards invite to callhandler and sends "100 Trying" response to loader.
    4. Callhandler responds to server with "180 ringing" response.
    5. Server forwards the 180 response to loader.
    6. Callhandler responds to server with "200 OK" response.
    7. Server forwards "200 OK" response to loader.
    8. Loader sends ACK to server; server forwards ACK to callhandler.
    9. Loader sends BYE to server; server forwards BYE to callhandler.
    10. Call handler responds to server with "200 OK" response for the BYE.
    11. Server forwards "200 OK" response to loader.

3. **SUBSCRIBE-NOTIFY**
    1. Loader sends SUBSCRIBE requests for presentities.
    2. Server stores subscription and responds with "200 OK" response.
    3. Server sends NOTIFY to loader; loader responds with 200 OK.
    4. Loader sends subscription refresh to server.
    5. Server renews subscription and responds with 200 OK response.

4. **PUBLISH-NOTIFY**
    1. Callhandler sends subscription for presentities to server.
    2. Server responds with 200 OK followed by NOTIFY for each presentity.
    3. Loader sends PUBLISH to server; server responds with 200 OK.
    4. Server composes notification and sends NOTIFY to callhandler.
    5. Callhandler sends 200 OK response to server.

Measurements and Results

Testing Environment

The testing was performed on IRT cluster machines. Each of the machines has Pentium 4 3 GHz CPU, on 800 MHz motherboard, with Redhat Linux version 2.6.9-22.0.2 and 1 GB of memory. The communication was over 100base-T Ethernet connection. There was minimal non-test traffic on the network.

The database server used was MySQL database version 4.1 running on `metro-north` (located on the same LAN as SUT) shared by all `sipd` and `pa` instances. However, this is not really an issue since the servers mostly use an in-memory database for storing requests and subscriptions.

Measurement

1. SIP server

For each of the tests, two test loaders and two test handlers were used. The number of users was fixed to 200 for each of the tests. The turn around time (TA) limit was set to 1000 ms for REGISTER tests and 2000 ms for INVITE tests. The acceptable success rate for each run is set to 50%. So if the success rate drops below this limit or the average TA goes above the specified limit, we assume that we have reached the maximum capacity of the server and stop further testing. The request step size used was 200 for UDP, 100 for TCP, and 40 for TLS. The test duration for each request step is set to 100 s.

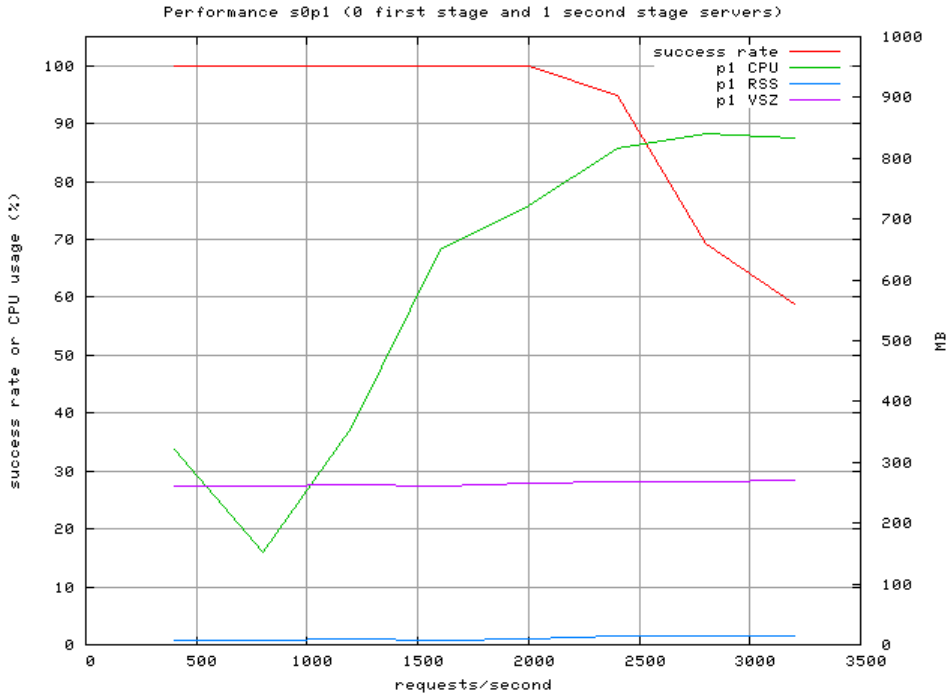


Figure 3: Register UDP with single server

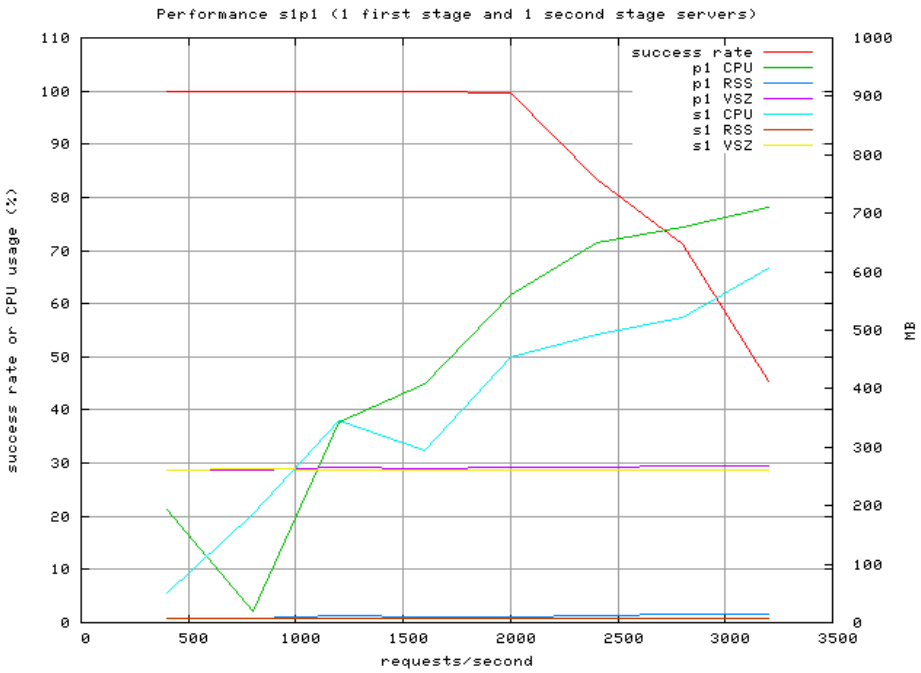


Figure 4: Register UDP with 1 stateless, 1 stateful server

The request handling capacity of the server with s0p1 configuration (figure 3) is 2000 req/sec. With s1p1 configuration (figure 4), there is no change in the request handling capacity of the server, since the only difference is that instead of getting the requests directly, the server gets the requests via a stateless proxy server.

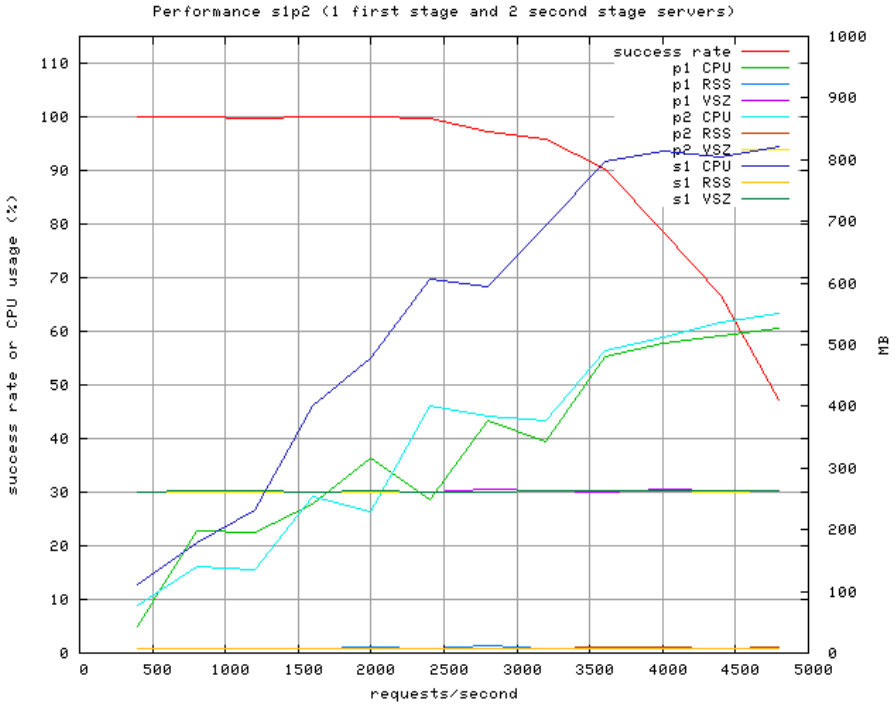


Figure 5: Register UDP with 1 stateless, 2 stateful servers

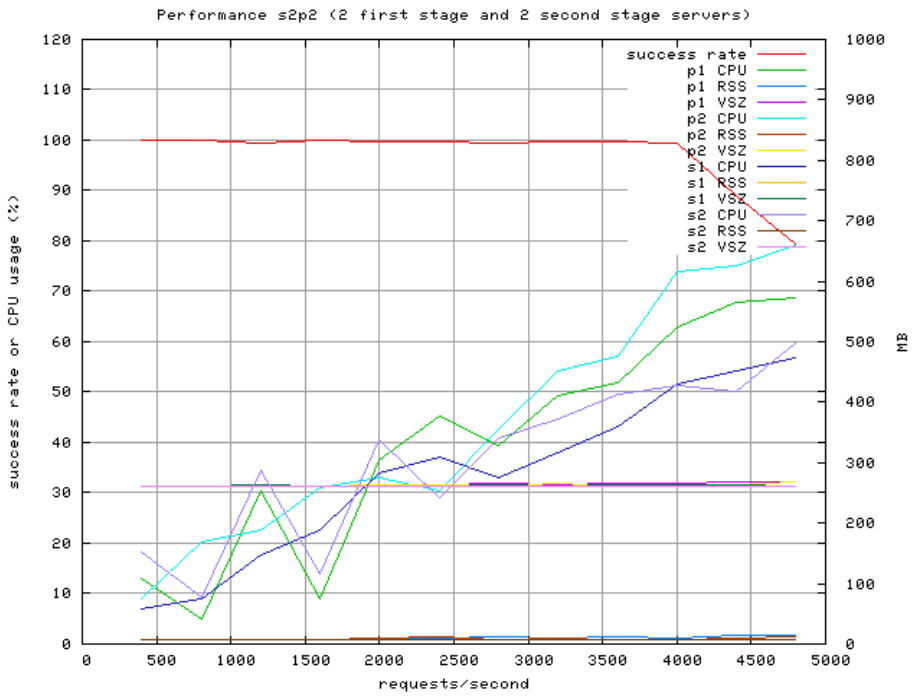


Figure 6: Register UDP with 2 stateless, 2 stateful servers

With s1p2 configuration (figure 5), the capacity of the server has increased to 2400 req/sec. The performance does not increase by 2-fold here by adding another stateful server because the performance bottleneck is the first stage stateless server, as seen by the CPU utilization in the graph (blue line). Thereby by adding another stateless server, the performance is expected to double that of s0p1 configuration. This is illustrated by the s2p2 configuration (figure 6) in which the capacity of the server increased to 4000 req/sec. Figure 7 shows a quick comparison with different load sharing configurations.

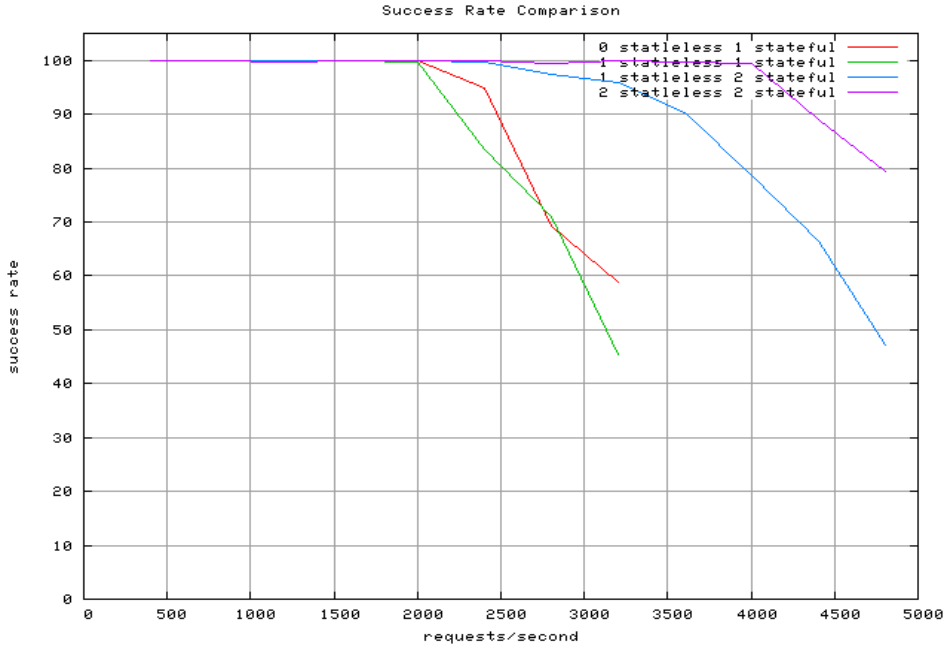


Figure 7: Register UDP with different load sharing configurations

Note that the CPU utilization reaches to 80-90% before the success rate starts dropping, and does not entirely go to 100% as expected. The reason is that we use 'ps' to compute CPU utilization of `sipd` on the server and not 'top', and since 'ps' gives aggregated CPU usage instead of instantaneous, it will be lower due to the initial wait period after starting `sipd`. This was verified by running 'top' on the hosts running `sipd` servers. If we increase the test duration to a considerably high value, say 400-500 sec for each request rate, we should be able to get the CPU utilization up to 100% in the graphs.

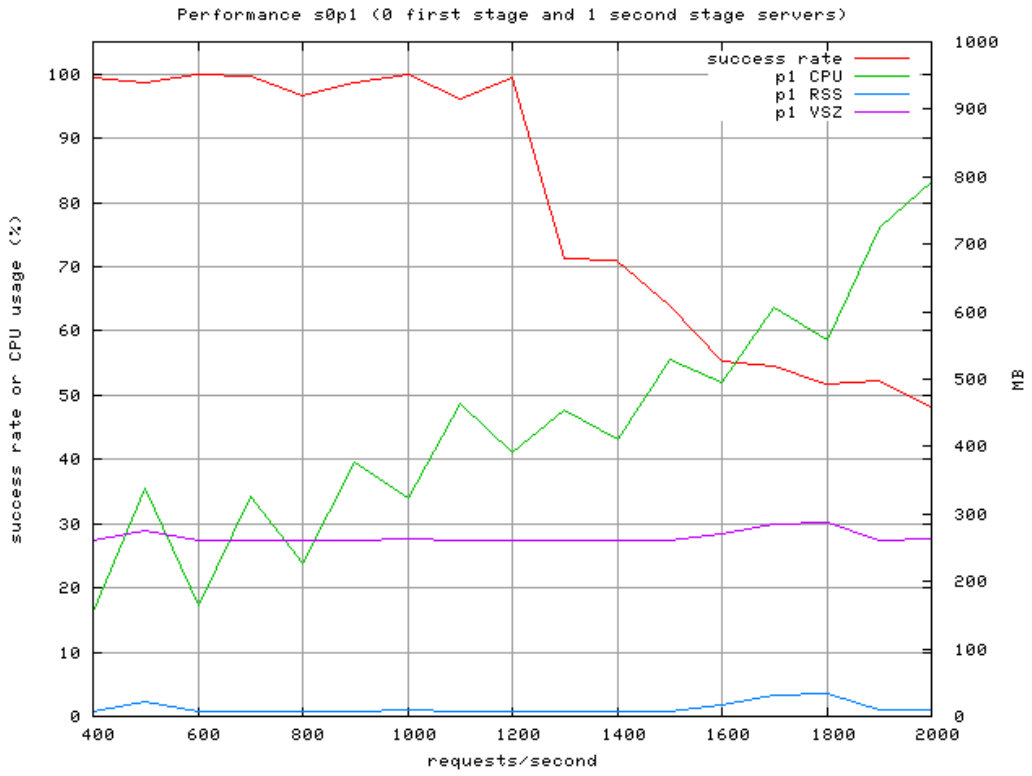


Figure 8: Register test with TCP

The request handling capacity of the SIP server for Register tests with TCP is around 1200 requests/sec. The scalability of SIP server for different load sharing configurations for TCP tests can be extrapolated from the UDP tests above.

Clearly TCP takes more memory than UDP as TCP requires maintaining state on the server. TCP has lower success rate than UDP on our test environment (where packet losses are negligible) because of the limit on the number of open connections on server. But when network packet losses are high or for certain mission-critical applications, TCP needs to be used as the transport protocol for SIP.

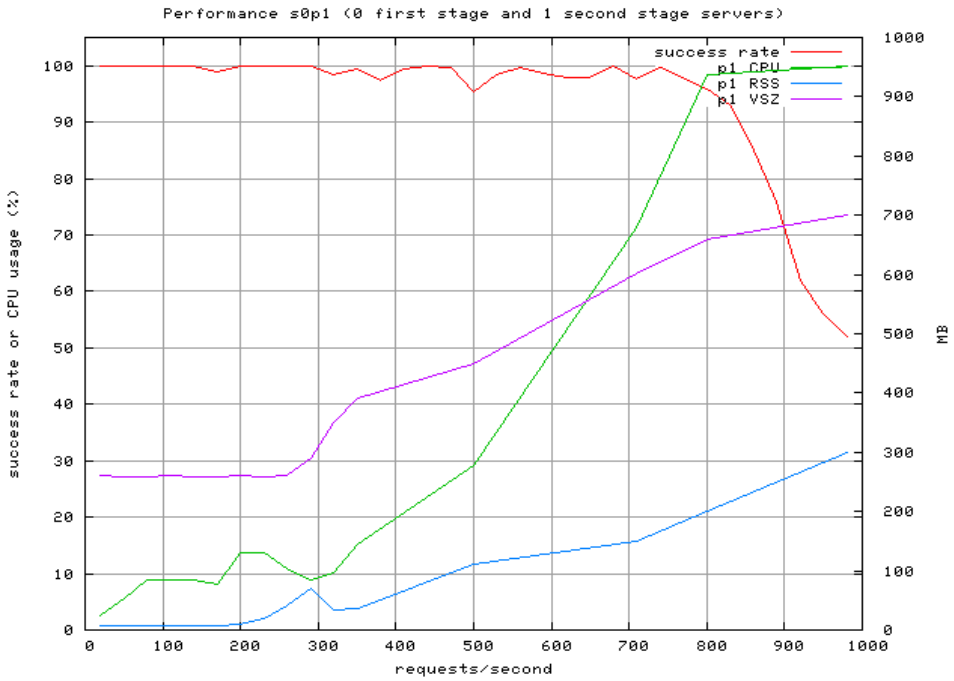


Figure 9: Register test with TLS

Figure 9 shows the performance of SIP server for Register with TLS. The request handling capacity of the server is roughly 720 req/sec. As we can see, the memory requirement for TLS is much greater than TCP or UDP because in addition to TCP state, the server also needs to maintain SSL session state for each TLS connection. As expected, TLS gives the lowest success rate, but needs to be deployed for security critical applications. Figure 10 gives a quick comparison between different transport protocols.

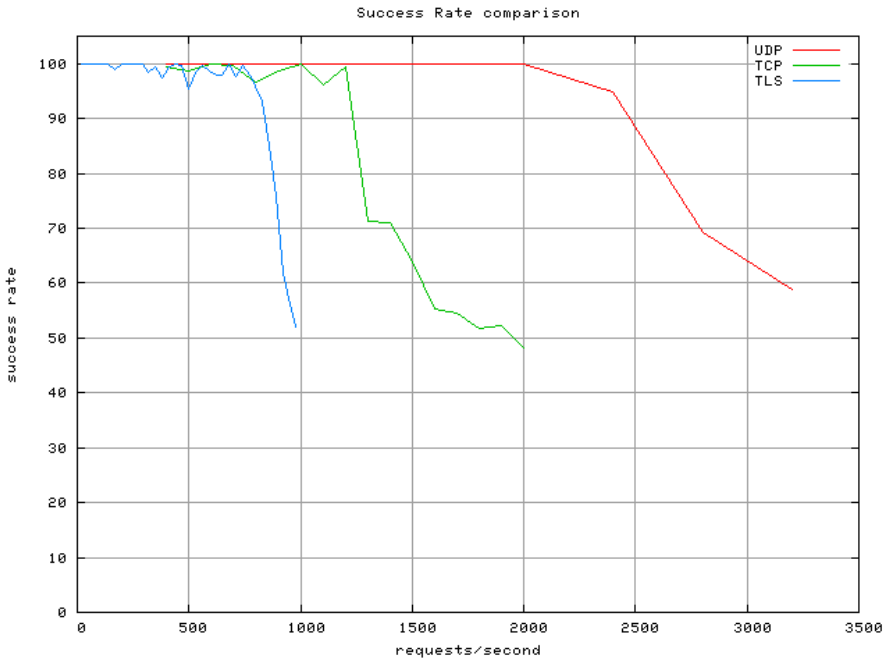


Figure 10: REGISTER with different transport protocols

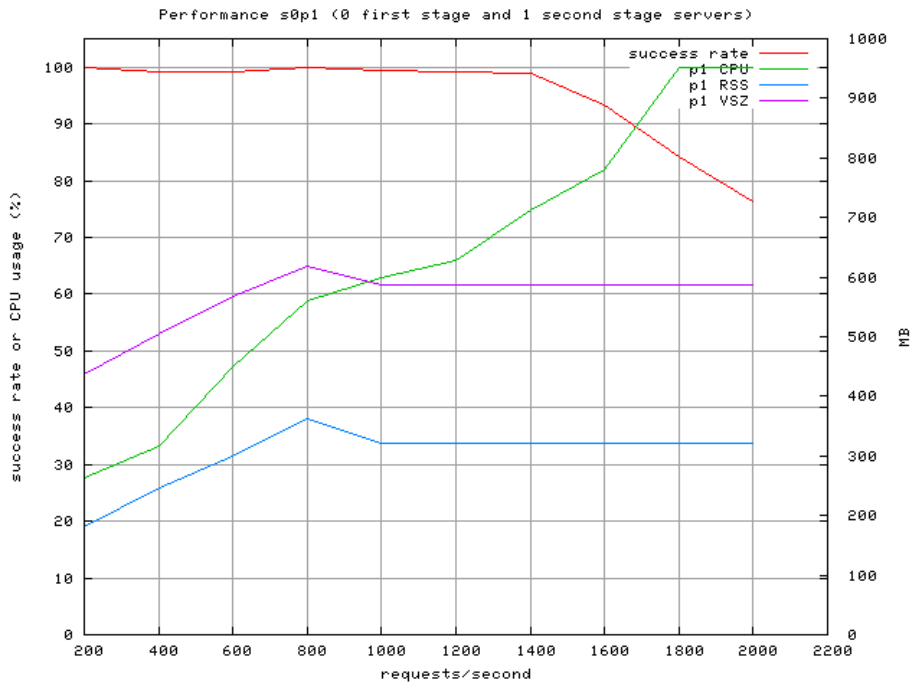


Figure 11: Invite test with UDP

Figure 11 shows the performance of SIP server for INVITE test with UDP. The request handling capacity is roughly 1400 req/sec as compared to 2000 req/sec for REGISTER test with UDP, since INVITE entails more server and network load, as discussed earlier. We can also see the big difference in the memory consumption between REGISTER and INVITE tests, since INVITE requires maintaining state on the server for each call, until the session is established.

The graphs for INVITE test with TCP and TLS are not shown here. The capacity of server for INVITE on TCP is 800 req/sec and with TLS is 400 req/sec. The drop in performance from TCP to UDP to TLS is significant for INVITE due to the requirement of maintaining connections/sessions on both sides of an INVITE call.

The performance comparison for INIVITE tests with different load sharing configurations can be extrapolated from REGISTER tests.

2. Presence Server Testing

Due to ongoing work on the presence server related to functional testing and performance optimizations, the performance test results are not available for presence server. However, they will be available in the near future.

Inconsistencies in results

1. In some cases, the CPU utilization does not reach 100% before the request rate starts dropping. The reasons for this are already explained in the previous section.

2. There are some spikes in the success rate graph for TCP and TLS testing. After comprehensive testing, we realize that this is not a problem with our benchmarking tool but due to some inconsistency in behavior of sipd indicating this might be a server implementation issue.

Future Work

- Presence server testing with different composition policies and size/type of subscriber and publisher filters.
- [1]To avoid unexpected and hard-to-debug errors, the wrapper scripts used for running sipstone tests should first create a test database, populate with relevant test data, run tests, and drop database. The temporary test database can be named according to user name and current timestamp. This avoids conflict with other tests and with other usages of the central database. Secondly, this will assist in seeing the impact of user population on performance of SIP servers. It should be easy to define a test database schema and test data, which can be updated whenever `sipd` or `pa` is modified.
- [1]Host name configuration used at various places such as domain DB table, certificate verification, sipstone, etc., should work with either canonical host name or IP address to avoid really hard-to-debug problems.
- [1]Implementation of DNS NAPTR/SRV schemes for lookups in sipstone.
- SSL session caching in SIP servers to avoid SSL handshake for every request.
- Connection reuse for SIP connections as specified in RFC draft [7].
- Add `transport` parameter to `contact` field in Register templates so that `sipd` can use the specified transport protocol for downstream calls, instead of the default.
- Changes in server code `sipd` need to be done in sync with `sipstone` so that new changes don't break the benchmark tool. Ideally the person responsible for some modification in `sipd` should make the corresponding changes in `sipstone`, so that the server is always ready for performance testing.

[1] this should definitely be done in the near future to improve usability of sipstone tools.

References

[1] SIPstone: Benchmarking SIP server performance
[2] SIMPLEstone: Benchmarking presence server performance.
[3] SIP redirect, proxy and registration server: sipd
[4] RFC 3261: SIP - session initiation protocol.
[5] RFC 2327: SDP - session description protocol.
[6] SIMPLE: SIP for instant messaging and presence.
[7] RFC Draft: Connection Reuse in SIP.
[8] Kundan Singh and Henning Schulzrinne, Failover and Load Sharing in SIP Telephony, Technical report, Department of Computer Science, Columbia University
[9] Jonathan Michael Lennox, Services for Internet Telephony, PhD Thesis, Columbia University, 2004

Appendix A. Program Documentation

1. Changes submitted to CVS.

2. Please see `cinema/sipstone/docs/sipstone.html` for revised documentation about running the tests.

3. *Testing with authentication:* First set the authentication of relevant test users to "required" in the test database. Also set the server authentication for your domain to "digest". The sql scripts *setauth.sql* and *setnoauth.sql* are provided to make this easy. Also for testing with authentication, set `useAuthentication` to true and set `preloadedCredentials` parameter.

4. *Testing with SSL:* We need to set up SSL keys and certificates for our test hosts signed with a common CA certificate in order to test the presence server with TLS. To automate this process, we have provided a script *generate_keystore.pl* that reads in a file "*hosts.txt*" and generates keys and certificates for each host in that file. It signs all certificates with a shared CA private key. If you are generating your own certificates, please make sure that the "CN" field in certificate is set to the canonical host name and NOT the IP address, since the server verifies by matching ip addresses. Also for testing with TLS, set the `protocol` to TLS, server port to 5061, and `security` to true in the test configuration.

5. Code changes in sipstone, sipd, reanalyze and wrapper test scripts submitted to CVS. Please see cinema/sipstone for revisions.

Appendix B. Examples

We have shown some examples below for different tests. Since the purpose here is to show the message flow, we have run these tests for 1 request, 1 user and 1 sec. duration.

1. Register UDP

test Loader	SUT
REGISTER sip:128.59.19.154 SIP/2.0 Via: SIP/2.0/UDP 128.59.19.159:7798 From: sip:A1000@cs.columbia.edu To: sip:A1000@cs.columbia.edu Contact: sip:A1000@128.59.19.159:7798 → Subject: sipstone register test CSeq: 1 REGISTER Call-ID: 1560721148@irtcluster07.cs.columbia.edu Expires: 86400 Content-Length: 0	SIP/2.0 200 OK Via: SIP/2.0/UDP 128.59.19.159:7798 From: sip:A1000@cs.columbia.edu To: sip:A1000@cs.columbia.edu; tag=1Lf7nhwQ6icuse8RcYVYsw Call-ID: 1560721148@irtcluster07.cs.columbia.edu CSeq: 1 REGISTER Date: Wed, 10 May 2006 08:25:42 GMT ← Server: Columbia-SIP-Server/1.24 Content-Length: 0 Contact: <sip:A1000@128.59.19.159:7798>; expires=86400; action=proxy; q=1.00 Expires: 86400

2. Register TCP

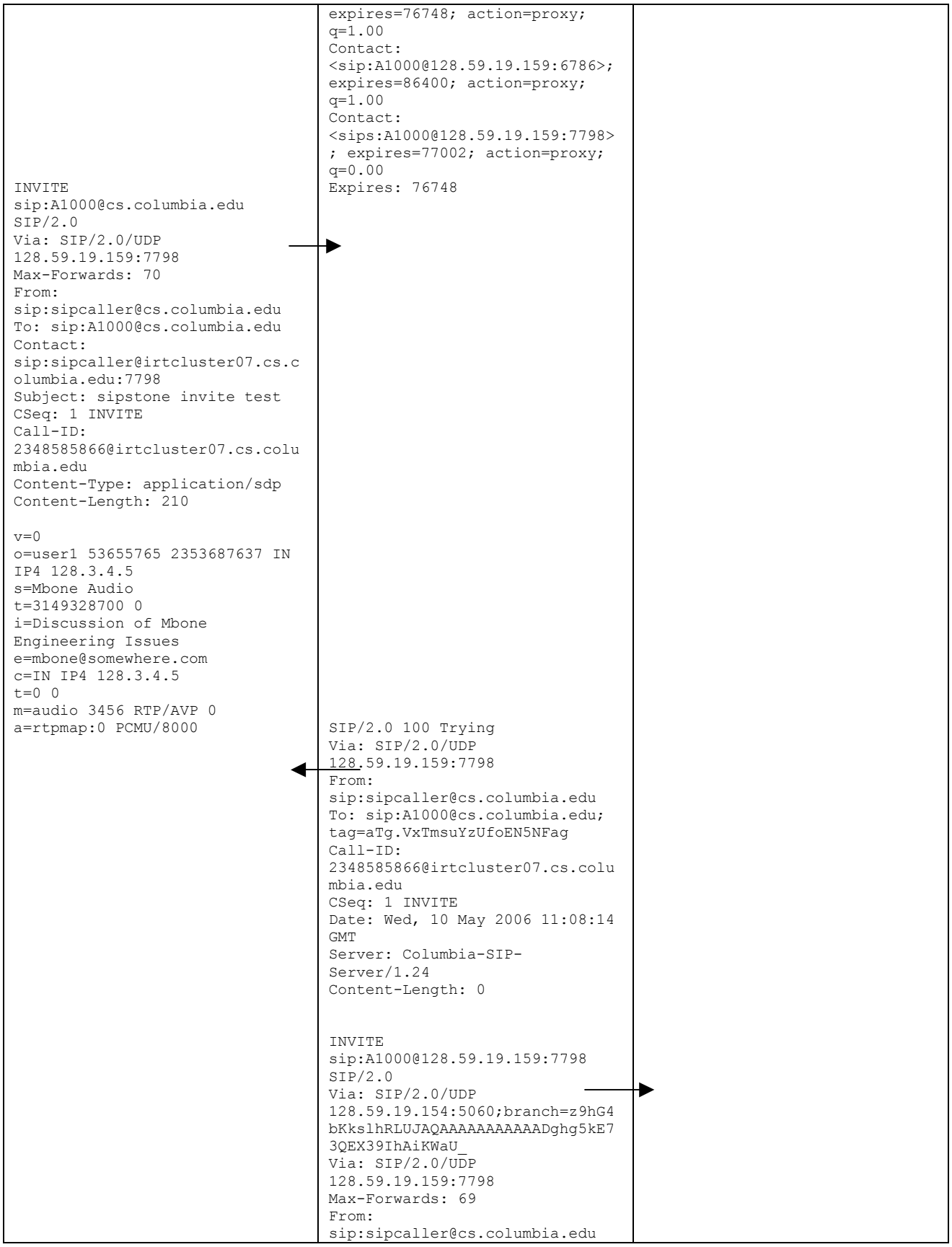
test Loader	SUT
REGISTER sip:128.59.19.154 SIP/2.0 Via: SIP/2.0/TCP 128.59.19.159:7798 From: sip:A1000@cs.columbia.edu To: sip:A1000@cs.columbia.edu Contact: sip:A1000@128.59.19.159:7798 Subject: sipstone register test → CSeq: 1 REGISTER Call-ID: 3181750771@irtcluster07.cs.columbia.edu Expires: 86400 Content-Length: 0	SIP/2.0 200 OK Via: SIP/2.0/TCP 128.59.19.159:7798 From: sip:A1000@cs.columbia.edu To: sip:A1000@cs.columbia.edu; tag=O9q6fXxde4lSZZZVnotj4Q Call-ID: 3181750771@irtcluster07.cs.columbia.edu CSeq: 1 REGISTER Date: Wed, 10 May 2006 08:27:19 GMT ← Server: Columbia-SIP-Server/1.24 Content-Length: 0 Contact: <sip:A1000@128.59.19.159:7798>; expires=86400; action=proxy; q=1.00 Expires: 86400

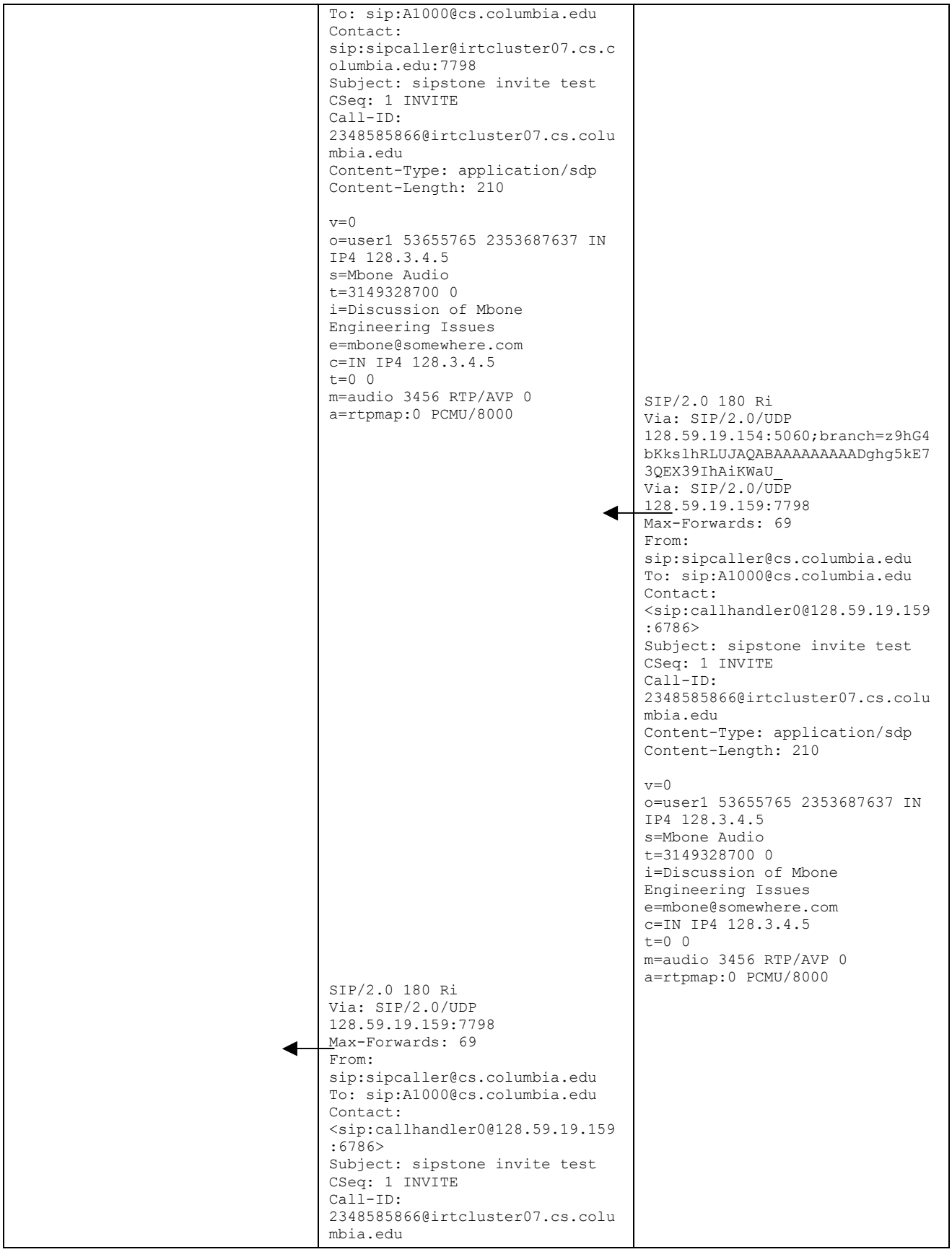
3. Register TLS

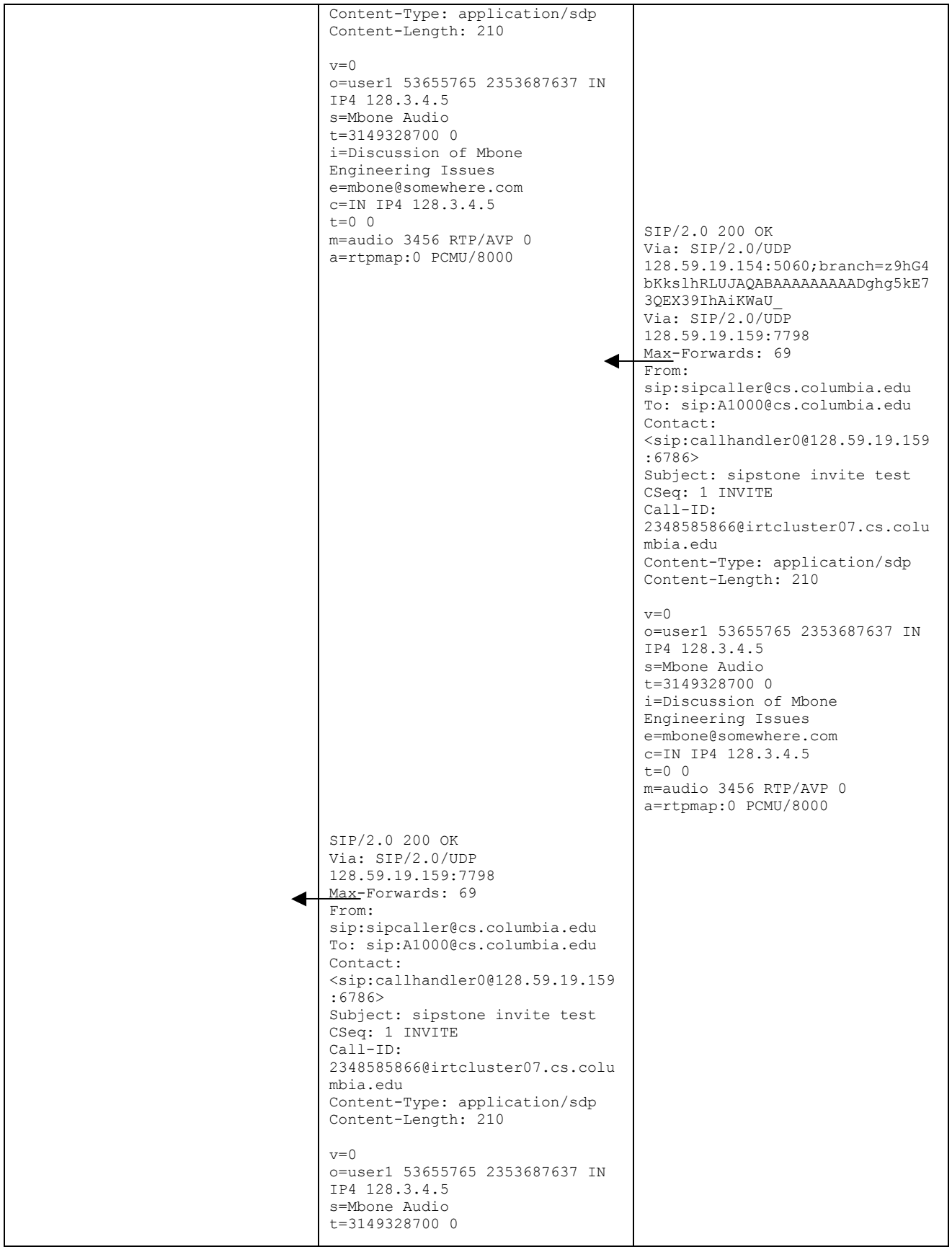
test Loader	SUT
REGISTER sip:128.59.19.154 SIP/2.0 Via: SIP/2.0/TLS 128.59.19.159:7798 From: sip:A1000@cs.columbia.edu To: sip:A1000@cs.columbia.edu Contact: sips:A1000@128.59.19.159:7798 Subject: sipstone register test CSeq: 1 REGISTER Call-ID: 239106195@irtcluster07.cs.columbia.edu Expires: 86400 Content-Length: 0	SIP/2.0 200 OK Via: SIP/2.0/TLS 128.59.19.159:7798 From: sip:A1000@cs.columbia.edu To: sip:A1000@cs.columbia.edu; tag=eF8jVurC85tFsYYE.0agjA Call-ID: 239106195@irtcluster07.cs.columbia.edu CSeq: 1 REGISTER Date: Wed, 10 May 2006 08:31:32 GMT Server: Columbia-SIP-Server/1.24 Content-Length: 0 Contact: <sip:A1000@128.59.19.159:7798>; expires=86146; action=proxy; q=1.00 Expires: 86146

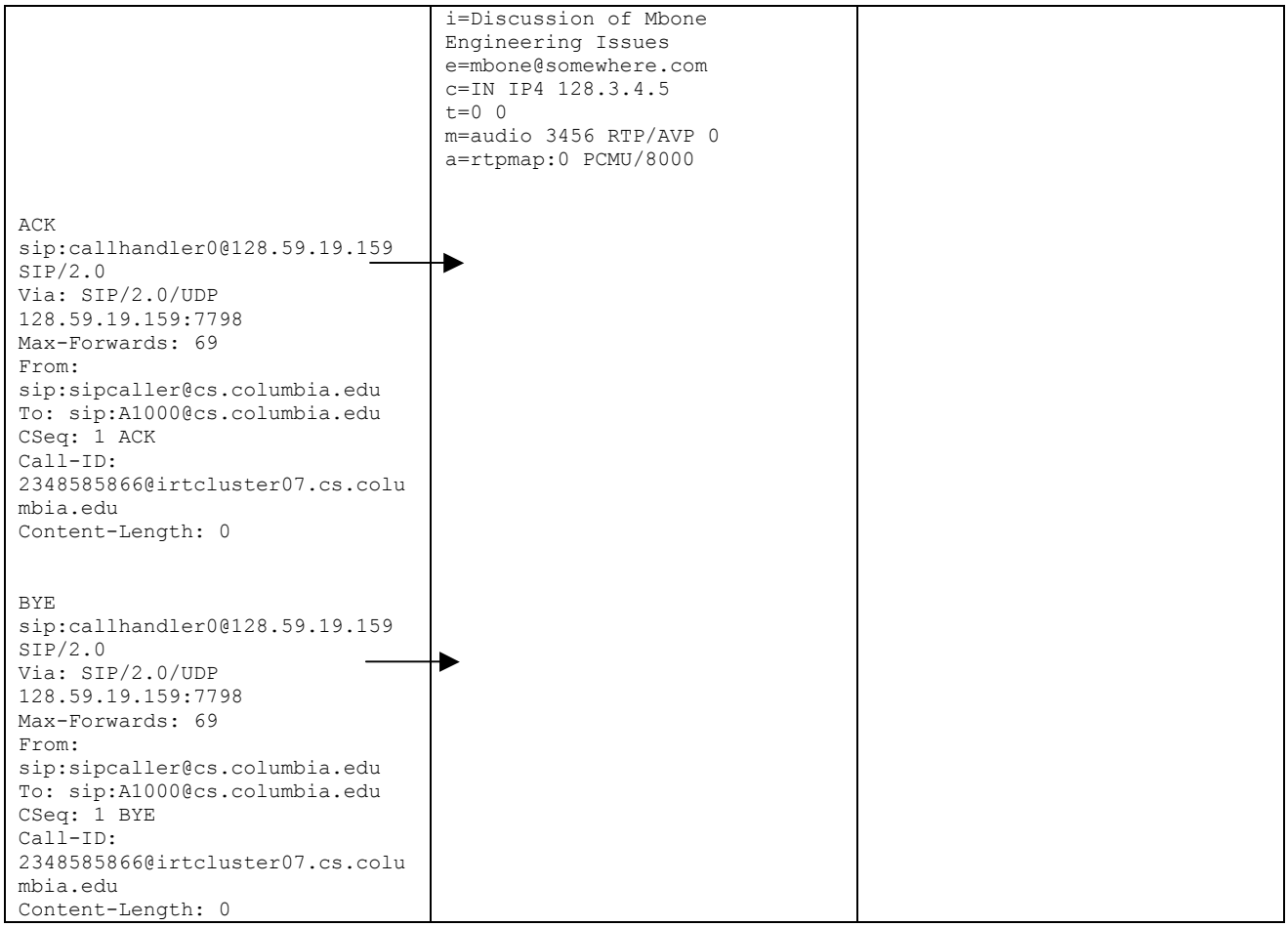
1. Invite UDP

test Loader	SUT	test Callhandler
		REGISTER sip:cs.columbia.edu SIP/2.0 Via: SIP/2.0/UDP 128.59.19.159:6786 From: sip:A1000@cs.columbia.edu To: sip:A1000@cs.columbia.edu Contact: sip:A1000@128.59.19.159:6786 Subject: sipstone register test CSeq: 1 REGISTER Call-ID: 646107955@irtcluster07.cs.columbia.edu Expires: 86400 Content-Length: 0
	SIP/2.0 200 OK Via: SIP/2.0/UDP 128.59.19.159:6786 From: sip:A1000@cs.columbia.edu To: sip:A1000@cs.columbia.edu; tag=gvwVveBPqSheRbH2LdxtnQ Call-ID: 646107955@irtcluster07.cs.columbia.edu CSeq: 1 REGISTER Date: Wed, 10 May 2006 11:08:11 GMT Server: Columbia-SIP-Server/1.24 Content-Length: 0 Contact: <sip:A1000@128.59.19.159:7798>;	


expires=76748; action=proxy;
q=1.00
Contact:
<sip:A1000@128.59.19.159:6786>;
expires=86400; action=proxy;
q=1.00
Contact:
<sips:A1000@128.59.19.159:7798>
; expires=77002; action=proxy;
q=0.00
Expires: 76748
INVITE
sip:A1000@cs.columbia.edu
SIP/2.0
Via: SIP/2.0/UDP
128.59.19.159:7798
Max-Forwards: 70
From:
sip:sipcaller@cs.columbia.edu
To: sip:A1000@cs.columbia.edu
Contact:
sip:sipcaller@irtcluster07.cs.c
olumbia.edu:7798
Subject: sipstone invite test
CSeq: 1 INVITE
Call-ID:
2348585866@irtcluster07.cs.colu
mbia.edu
Content-Type: application/sdp
Content-Length: 210
v=0
o=user1 53655765 2353687637 IN
IP4 128.3.4.5
s=Mbone Audio
t=3149328700 0
i=Discussion of Mbone
Engineering Issues
e=mbone@somewhere.com
c=IN IP4 128.3.4.5
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
SIP/2.0 100 Trying
Via: SIP/2.0/UDP
128.59.19.159:7798
From:
sip:sipcaller@cs.columbia.edu
To: sip:A1000@cs.columbia.edu;
tag=aTg.VxTmsuYzUfoEN5NFag
Call-ID:
2348585866@irtcluster07.cs.colu
mbia.edu
CSeq: 1 INVITE
Date: Wed, 10 May 2006 11:08:14
GMT
Server: Columbia-SIP-
Server/1.24
Content-Length: 0
INVITE
sip:A1000@128.59.19.159:7798
SIP/2.0
Via: SIP/2.0/UDP
128.59.19.154:5060;branch=z9hG4
bKkslhRLUJAQAAAAAAAAAADghg5kE7
3QEX39IhAiKWaU_
Via: SIP/2.0/UDP
128.59.19.159:7798
Max-Forwards: 69
From:
sip:sipcaller@cs.columbia.edu


To: sip:A1000@cs.columbia.edu
Contact:
sip:sipcaller@irtcluster07.cs.c
olumbia.edu:7798
Subject: sipstone invite test
CSeq: 1 INVITE
Call-ID:
2348585866@irtcluster07.cs.colu
mbia.edu
Content-Type: application/sdp
Content-Length: 210
v=0
o=user1 53655765 2353687637 IN
IP4 128.3.4.5
s=Mbone Audio
t=3149328700 0
i=Discussion of Mbone
Engineering Issues
e=mbone@somewhere.com
c=IN IP4 128.3.4.5
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
SIP/2.0 180 Ri
Via: SIP/2.0/UDP
128.59.19.154:5060;branch=z9hG4
bKkslhRLUJAQABAAAAAAAAADghg5kE7
3QEX39IhAiKWaU_
Via: SIP/2.0/UDP
128.59.19.159:7798
Max-Forwards: 69
From:
sip:sipcaller@cs.columbia.edu
To: sip:A1000@cs.columbia.edu
Contact:
<sip:callhandler0@128.59.19.159
:6786>
Subject: sipstone invite test
CSeq: 1 INVITE
Call-ID:
2348585866@irtcluster07.cs.colu
mbia.edu
Content-Type: application/sdp
Content-Length: 210
v=0
o=user1 53655765 2353687637 IN
IP4 128.3.4.5
s=Mbone Audio
t=3149328700 0
i=Discussion of Mbone
Engineering Issues
e=mbone@somewhere.com
c=IN IP4 128.3.4.5
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
SIP/2.0 180 Ri
Via: SIP/2.0/UDP
128.59.19.159:7798
Max-Forwards: 69
From:
sip:sipcaller@cs.columbia.edu
To: sip:A1000@cs.columbia.edu
Contact:
<sip:callhandler0@128.59.19.159
:6786>
Subject: sipstone invite test
CSeq: 1 INVITE
Call-ID:
2348585866@irtcluster07.cs.colu
mbia.edu


Content-Type: application/sdp
Content-Length: 210
v=0
o=user1 53655765 2353687637 IN IP4 128.3.4.5
s=Mbone Audio
t=3149328700 0
i=Discussion of Mbone Engineering Issues
e=mbone@somewhere.com
c=IN IP4 128.3.4.5
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
SIP/2.0 200 OK
Via: SIP/2.0/UDP 128.59.19.154:5060;branch=z9hG4bKkslhRLUJAQABAAAAAAAAADghg5kE73QEX39IhAiKWaU_
Via: SIP/2.0/UDP 128.59.19.159:7798
Max-Forwards: 69
From: sip:sipcaller@cs.columbia.edu
To: sip:A1000@cs.columbia.edu
Contact: <sip:callhandler0@128.59.19.159:6786>
Subject: sipstone invite test
CSeq: 1 INVITE
Call-ID: 2348585866@irtcluster07.cs.columbia.edu
Content-Type: application/sdp
Content-Length: 210
v=0
o=user1 53655765 2353687637 IN IP4 128.3.4.5
s=Mbone Audio
t=3149328700 0
i=Discussion of Mbone Engineering Issues
e=mbone@somewhere.com
c=IN IP4 128.3.4.5
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
SIP/2.0 200 OK
Via: SIP/2.0/UDP 128.59.19.159:7798
Max-Forwards: 69
From: sip:sipcaller@cs.columbia.edu
To: sip:A1000@cs.columbia.edu
Contact: <sip:callhandler0@128.59.19.159:6786>
Subject: sipstone invite test
CSeq: 1 INVITE
Call-ID: 2348585866@irtcluster07.cs.columbia.edu
Content-Type: application/sdp
Content-Length: 210
v=0
o=user1 53655765 2353687637 IN IP4 128.3.4.5
s=Mbone Audio
t=3149328700 0



The examples for Invite TCP and Invite TLS are similar to Invite UDP shown above.