SipStone - Benchmarking SIP Server Performance

Abstract

The ability to compare the performance of different SIP Telephony Server configurations from vendors is critical in selecting a telephony server. To compare different servers, we need a benchmark algorithm and a result collation mechanism. This document presents one such benchmark for evaluating SIP Proxy, Redirect and Registrar servers. It is to be noted that this document is in draft form.

I. INTRODUCTION

SipStone is a benchmark for SIP (Session Initiation Protocol [1]) Proxy and Redirect Servers (SIP Server). SipStone attempts to measure the performance of a SIP Server in terms of its request handling capacity. It is concerned with the overall response time and treats the SIP Server as a black box.

SipStone is designed to be a benchmark with a single standardized implementation and workload. The implementation performs a series of tests that generate the pre-configured workload. This workload is performed in a controlled IP Telephony test bed that simulates the activities of multiple users initiating SIP Calls. The workload is intended to be simple and repeatable, and exercises a breadth of system components associated with this environment such as concurrent initiation of calls by multiple users, forwarding of calls by SIP Proxies etc.

The remainder of this document is organized as follows: Section II presents an overview of SIP Servers, and illustrates SIP call setup with several examples. Issues involved in designing a benchmark for an Internet Telephony system are discussed in Section III. Section IV presents the architectural features of SipStone such as the test bed components and load generation mechanisms. Performance metrics for SIP Servers are defined in Section V. Section V also details the measurement methodology used for benchmarking. Section VI discusses issues that are to be addressed in the next revision. Section VII presents conclusions.

II. OVERVIEW OF SIP SERVERS

Before we look at benchmarking mechanisms it is helpful to know the categories of SIP Servers, and their operation. Readers familiar with SIP may skip this section. SIP supports user mobility by allowing addresses in the form of *user@domain* to be used when making Internet audio calls. For instance *alice@home.com* can call *bob@office.com* no matter what communication device, IP address or phone number bob is using currently.The locations where users could be reached are maintained by the SIP location or registration servers.

When Alice, with address *sip:alice@home.com*, wants to call Bob, *sip:bob@office.com*, her SIP phone contacts the server at *office.com*. The server knows where Bob can be reached and can either return Bob's location to Alice's phone (the redirect mode) or can itself try to contact Bob at his current location (the proxy mode). In the former case, Alice's phone retries the new location, while in the latter case the server proxies the request transparent to the caller. It is possible to encounter multiple SIP servers (either in redirect or proxy mode) in a given call attempt. A *forking proxy* can fork the call request to more than one locations, so that the first phone that is picked-up gets through the call, while all the other phones stop ringing.

The list of audio/video algorithms supported and the transport addresses to receive them, are described using Session Description Protocol (SDP [2]), carried in SIP requests and responses.

III. ISSUES IN DESIGNING A BENCHMARK

The goal of any benchmarking system is to help understand how a system will perform, and to help compare different implementations. The performance of a SIP server is dependent not only on the implementation but also on factors like size of the user population serviced, and the nature of requests submitted. By nature of requests, we mean their arrival and service time distributions. Using an accurate model of the real workload in benchmarking, increases the value of the benchmark. Additionally, factors such as the transport protocol used, forking by proxy servers, DNS Vs. IP addressing should also be considered. We elaborate on each of these issues in the following subsections.

A. User Population

The performance of a SIP Server is dependent in addition to many others on the size of the user population making use of the server. This is because, a typical SIP transaction involves looking up the contact addresses of a given user. Suppose for example, a SIP Registrar server uses a database to store user information including contact addresses. High performance could be obtained by maintaining the contact addresses of a couple of users in the main memory, and performing database queries only when the data is not present in main memory. As the number of users increase, the probability of an item in main memory decreases. This implies that a server performing well with a low user population, or a user population with a large number of inactive users, scales poorly as the user population grows. A good benchmark takes these considerations into account when specifying the workload mix.

B. *Request Modeling*

Teletraffic theory [4] provides statistical models for arrival and service time distributions of calls at PSTN switches. These models can be used as a preliminary basis for modeling workload generation. A brief overview of the relevant models is presented below.

Traditionally, a possion process is used to characterize PSTN call arrivals [4]. Properties of the possion process that make it well suited to model call arrivals include stationarity and independent increments. The stationarity implies that the call arrivals in two non-overlapping time intervals is independent. However, empirical analysis with actual CCSN [3] traffic data have shown that during a 24-hour period, call arrivals are high during day times, and less during night, thus invalidating the stationarity assumption. A time-inhomogenous possion process accounts for these variations, and has been shown citeccsn to model the emprical call arrivals quite accurately.

The time between setup and teardown is the conversation time or the call holding time. The exponential distribution is the traditional method of approximating call holding times in a switch. Recent studies [3] have shown that call holding times in PSTN switches are more accurately modeled by a heavy-tailed distribution.

C. *Standard Call Transaction*

Measuring the response time for a request does not present a correct picture of the actual performance. For example, consider sending an INVITE request to a proxy server that needs to be forwarded by the proxy to a UAC. Measuring the elapsed time between sending the INVITE and receiving a 100 Trying skews the service time distribution. On the other hand, if the time to receive a 100 Trying is of the order of a few seconds, then the initiating client will perform several retransmissions according to retransmit rules. This increases the load on the server. Hence, in the request mix used to benchmark a server, we need to consider the response time of a full transaction such as INVITE, [100 Trying, 180 Ringing], 200 OK.

D. *Other Issues*

Other SIP specific issues include the use of different transport protocols such as TCP and UDP. Clearly, separate tests should be used for UDP and TCP. When using TCP, issues such as the ability of the server to handle a large number of open connections, use of persistent connections need to be considered. When UDP is used, the effect of packet losses and retransmissions should be taken into account.

Mechanisms such as forking and Via-header processing impose higher processing demands on the Proxy Server. Forking in SIP Proxy Servers was discussed in SectionII. When a chain of proxy servers are used, handling of the Via header becomes critical in processing the message. The workload should be able to make the proxy server perform such computations. In this version of this document, we do not consider forking and Via-header processing. In addition to the above, size of the SDP used with SIP requests has the potential to affect performance. A large SDP may result in several datagrams when UDP is used.

Other relevant issues to consider when designing a benchmark include repeatability, measurability, and scalability. Repeatability refers to the ability to replicate the results by a third-party when running the test under the specified environment. Measurability refers to the ease with which the measured test results can be quantified in a way to present a picture of the system being measured. The ability of the benchmark to run on different configurations such as simulating a large number of users, different arrival distributions refers to scalability. We elaborate on these issues further in the next section.

IV. SIPSTONE

SipStone is an architecture for generating workload, a workload specification based on SIP, a suite of tests that exercise the various components of the system being tested, rules and metrics to perform the tests and interpret results.

A. *Architecture*

The Server under test (SUT) is a SIP Proxy and/or Redirect server whose performance is to be estimated. SipStone benchmark consists of a set of SipStone load generators that create the SIP request load, a call handler that simulates a user agent client, a central SipStone manager (coordinator) that coordinates the execution of the benchmark, and the SUT. Actual measurements are based on the controlled workload profile described in Section V. The call handlers may run along with the load generators or on different systems as shown below. Fig. 1 shows an example test system with four load generators distributed over two LAN segments making call requests to a single server.

A.1 Server under test

The SUT consists of the host system(s) (including hardware and software) required to support the SIP Telephony server and any other components including database(s) if applicable. All network components (hardware and software) between host machines, which handle intra-SUT application communications, are part of the SUT. The software consists of the SIP Redirect and/or Proxy server. This software is referred to as the SIP Server in the ensuing discussion. For simplicity, this document considers the registrar server to be co-located with the SIP Server.

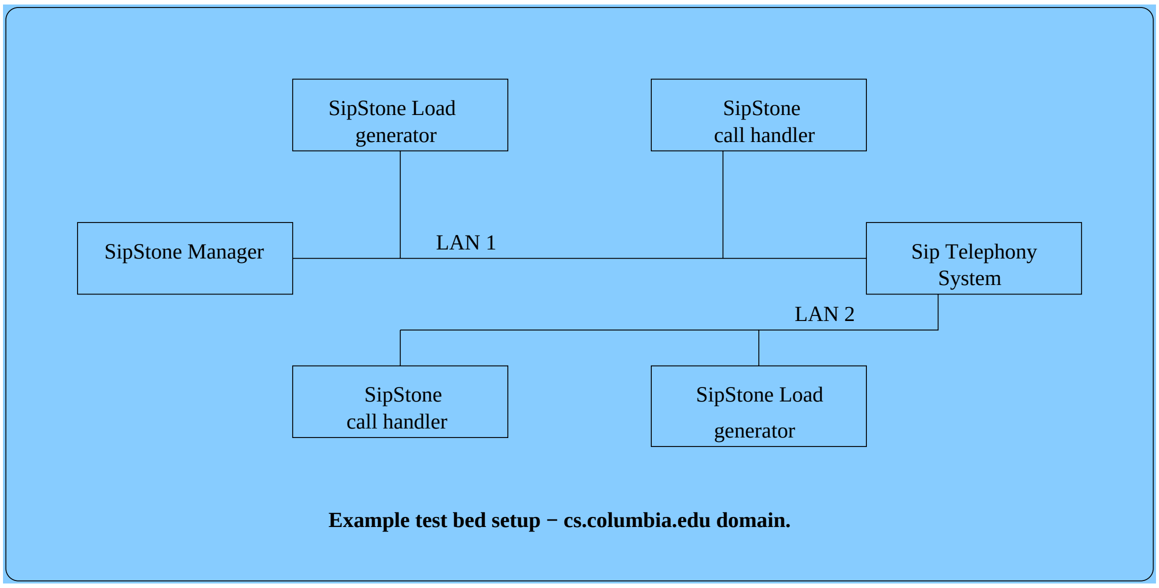

Fig. 1. Architecture

A.2 Clients

SipStone executes a set of cooperating distributed processes. A master program reads the input benchmark configuration file. The benchmark configuration file contains details about participating hosts, the number of users to be simulated by each host, parameters for arrival distribution etc., The master program invokes the load generator on each of the participating computers using mechanisms such as rsh/rcmd, RMI activation etc., The load generator spawns a number of workers to create the generated load based on the specified configuration, sets up log and error files. The master program also starts call handlers on participating computers. Each worker is capable of generating SIP requests with possion distribution. The mean of the possion distribution, and the type of the SIP request is specified as configuration parameters.

The coordinator controls the various phases of the benchmark and consolidates the performance results from each load generator. SipStone workers implement the SIP Protocol within the benchmark. This design enables the workers to use either TCP or UDP for SIP Transport. It also helps validate the response. On receipt of a response, each worker records the response time, and the result of executing the test (successful or failure). If a proxy server is tested, then the call handler is used to handle proxied calls. For the sake of simplicity, SipStone workload profiles involve calls that involve only two call-legs.

A.3 Call Handler

A typical action carried out by a SIP Proxy server is forwarding a call to the Callee transparently to the Caller. To test this scenario using SipStone, we need a component that simulates the actions of a Callee. The Call Handler component is used for this purpose. On receipt of an incoming call, the SIP Proxy server decides whether to forward the call based on the contact locations registered by the Callee. The Call Handler component simulates the Callee and responds to the request.

B. Generating SIP requests

During startup, the total user population is divided equally among the configured call handlers. Call handler's register the users they represent with the SUT. When the test is carried out, each SipStone worker picks the Caller and Callee(if applicable) for the SIP request randomly from the user population based on a uniform distribution. This has the effect of simulating a large active user population. Scaling guidelines for the number of users are given in Section V.

The SIP requests used for benchmarking can be divided into INVITE and REGISTER. The workload profile is a mix of INVITE and REGISTER transactions that generate final responses as given in Table I. Since proxy and redirect servers are different functional entities, the workload mix is designed to accommodate the differences in function.

Three different interaction scenarios relevant to the benchmark suite are given below. Only the high-level message flow is described here. Exact contents of the SIP request and response message for these interactions can be found in [5].

In Figure 2, *User A* is simulated by one of the workers. User A is registered during the beginning of the test with the SUT. The registration is refreshed at periodic intervals as shown. A register call transaction includes all the messages from the starting REGISTER method, till the 200 OK is received. We do not consider authentication in this document, and hence, no authentication credentials are supplied with the registration.

Figure 3 describes the typical action of a redirect server. When a SIP INVITE Request is received by the redirect server, it sends

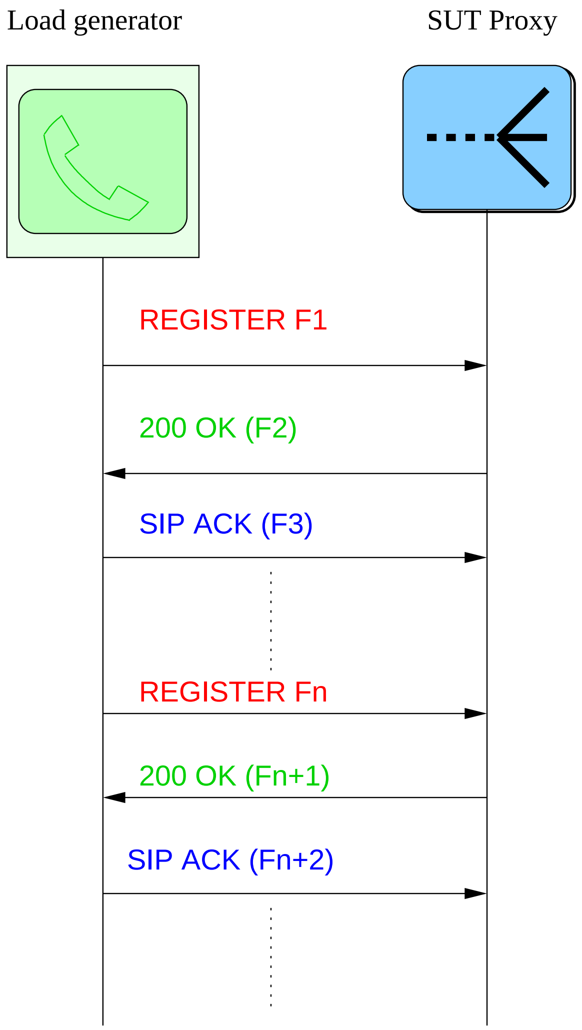

Fig. 2. Scenario I. REGISTER with periodic refresh

a 3xx response redirecting the INVITE request. All messages from the initial INVITE to receiving a 3xx response constitute a call transaction.

Typical operational scenario for proxy servers is given in Figure 4. Here, User A completes a call to User B using a proxy Proxy 1. As mentioned before, the proxy is non-forking. To simulate this scenario in SipStone, User A is represented by the load generator. User B is represented by a SipStone call handler, which is registered with the Proxy Server. The transaction completes when User A receives a final response. After a configured think time, User A sends a BYE requests to tear down the session.

V. BENCHMARKING METHODOLOGY

SipStone benchmarking consists of a series of test runs, with increasing load levels generated by the load generators, and targeted at the SIP Server being tested. Benchmark configuration parameters should be chosen based on the requirements given in the following subsections. The following configuration parameters are constant over a test run:

- All aspects of server hardware and software configuration are fixed over the benchmark run.
- number and type of SipStone load generators that participate in the run, and the number of worker threads created in each generator.
- number and type of network segments connecting the load generators to the SIP server.
- the arragement of load generators on the network segments.
- the workload parameters except the load level, which may be increased across several test runs.

In the following subsections we present the benchmarking scheme for SIP Servers, and present two metrics *Calls Per Second, and* $/CPS. We discuss the requirements to be met in a benchmarking environment, and detail the methodolgy for computing the metric from the data collected from test runs.

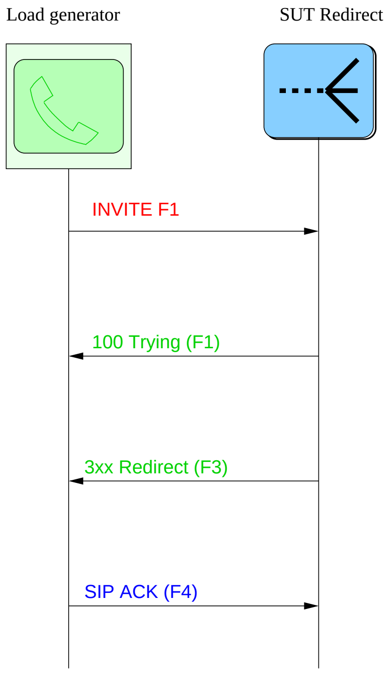

Fig. 3. Scenario II. INVITE redirected

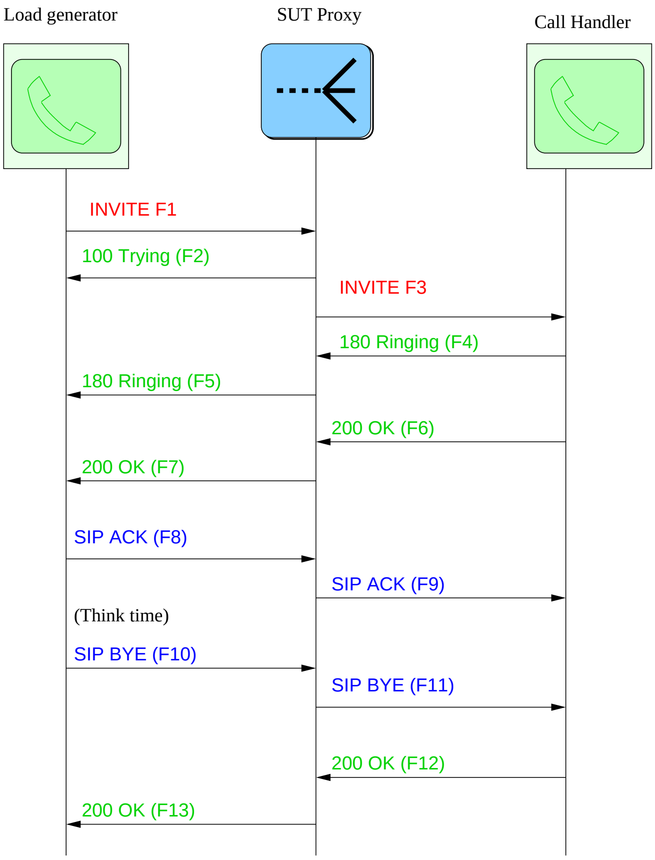

Fig. 4. Scenario III. INVITE with call forwarding

A. *Definition of Terms*

Startup Interval (SI) : Startup Interval refers to the time (in seconds) in which the load generators are initialized. During this interval, workers are started, and they perform any necessary initializations. Call handlers also register the SIP Users they represent with the SUT. At the end of startup interval, all the components are ready to start load generation.

Warmup Interval (WI) : Warmup Interval refers to the time (in seconds) needed to get to steady state prior to the actual start of measurements. This may be needed to warm up any caches before starting measurements.

Measurement Interval (MI) : Measurement Interval is defined as the steady state period during the execution of the benchmark from which the reported performance metric is derived. During the measurement interval the workers generate sip requests, and carry out the test.

Call Response Time (CRT) : Call Response Time is defined as the time elapsed from the first byte sent by a worker to initiate a call until the last byte received by the worker that ends the transaction.

CPS (Calls per second) : Calls per second is defined as the average number of successful calls per second completed during a measurement interval.

Cost per CPS ($/CPS) : Cost per CPS is defined as the total cost of the SUT divided by the computed CPS. The total cost of the SUT is the cost of SUT hardware, software and network components used to achieve the reported CPS. It does not include the cost of client components used in testing.

CRS (Call reject per second) : Call reject per second is defined as the average number of call initiation requests that were rejected during a measurement interval. This includes only errors that occurred due to server software limitations that resulted in client getting a timeout, a 5xx error (to complete).

Scenario	Response	Description	Applicability
Figure 2	non-1xx	Measuring lookup time	Redirect and Proxy
Figure 3	non-1xx	Typical scenario	Redirect only
Figure 4	non-1xx	Involves call forwarding	Proxies only

TABLE I

WORKLOAD PROFILES.

B. Measurement Methodology

A Test Run consists of one or more Measurement Intervals. Each Measurement Interval is preceded by a ramp-up period during which the Startup and Warmup phases are completed. At the end of this ramp-up period, a steady state load generation level is reached. During the steady state interval, the workers generate a continuous stream of sip requests based on the profile given in Fig. I. The workers generate possion distributed requests and measure the response time. When the optimal operating capacity of the server is found, the measurement is repeated three times to compute the performance metric.

C. Scaling requirements

The size of the user population used for the tests should scale with the request handling capacity of the server.

D. Response time constraints

During each Measurement Interval, at least 90% of call setup requests of each type must have a CRT of less than the constraint specified (in milli seconds) as per Table II. These values are designed such that they are less than the retransmission default T1 timer of 500 ms. Additionally, since these tests will be typically carried out in a LAN, the values specified are much smaller. Instead of triggering a retransmission when the timer elapses, the worker marks it as failure and continues with the next iteration. The average CRT of a measurement interval is computed by averaging over the individual response times that fall within the 90th percentile for that interval.

Scenario	Response	Response Time (ms)	Description
Figure 2	400	200	Final response before retransmission occurs
Figure 4	1xx	100	Initial 1xx (if applicable) in all scenarios
Figure 3	non-1xx	400	For redirect servers, before retransmission occurs
Figure 4	non-1xx	2000	For a simple SIP call through the proxy

TABLE II

ALLOWED RESPONSE TIMES.

E. Throughput computation

The throughput (CPS) for each of the measurement intervals is computed as the total successful transactions in that interval divided by length of the interval in seconds. Here total successful transaction is defined as the fraction of the total transactions that generated a valid 2xx, 3xx, or 4xx response and whose response time falls within the average CRT of the measurement interval. The reported throughput must be computed over a Measurement Interval during which the throughput level is in a steady state condition that represents the true sustainable performance of the SUT. The final reported CPS and $/CPS metrics are based on the average throughput over three measurement intervals.

F. Locating the optimal operating point

To locate the optimal operating point at which the throughput is optimal, the experiment has to be first reperated for varying values of the mean of the arrival distribution. Plotting the actual throughput for varying arrival rates, and extrapolating them helps to find the correct arrival rate which maximizes performance.

G. Metrics and Parameters to be reported

When declaring the results, these are some of the data that need to be provided.

- The number of connections requested by the clients and accepted by the SUT per second. The intent is to count only the number of new connections made successfully by the clients in generating the load for the benchmark.
- CPU and memory utilization of server.
- Page swap activity.
- System i/o activity.
- The number of concurrent requests being handled by the server / second.
- CRS.
- Log Data of the server.

VI. LIMITATIONS

The current draft does not provide the procedural model that needs to be used to find the correct operating point. It also does not specify the configuration, reporting, and logging requirements. These will be addressed in the next revision.

VII. CONCLUSIONS

A preliminary benchmarking scheme for SIP Telephony servers was presented. We discussed several issues to be considered while designing servers, and presented an architecture and a technique to measure Server performance.

REFERENCES

[1] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, “SIP: session initiation protocol,” Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[2] M. Handley and V. Jacobson, “SDP: session description protocol,” Request for Comments 2327, Internet Engineering Task Force, Apr. 1998.

[3] D. E. Duffy, A. A. McIntosh, M. Rosenstein, and W. Willinger, “Statistical Analysis of CCSN/SS7 Traffic Data from Working CCS Subnetworks,” in *IEEE Journal on Selected Areas in Comminications*, 12(3), Apr. 1994.

[4] J. Bellamy, *Digital Telephony*. New York: John Wiley & Sons, 1991.

[5] A. Johnston, et al., “SIP Call Flow Examples,” Internet Draft, Internet Engineering Task Force, Apr. 2001.