IMS Bench SIPp

Introduction

by David Verbeiren (Intel), Philippe Lecluse (Intel), Xavier Simonart (Intel)

Table of contents

1 Overview

The "IMS Bench SIPp" is a modified version of SIPp with supporting scenario files and a few tools meant to provide an open source implementation of a test system conforming to the IMS/NGN Performance Benchmark specification, ETSI TS 186 008.

The modifications made to SIPp include many new features that were required in order to implement the IMS benchmark specification and to apply the resulting test system to potentially large IMS systems. These features and changes are detailed in a later section.

The scenario files currently provided cover the following scenarios from the specification:

- Successful call
- Successful messaging
- Registration
- De-registration
- Re-registration

A report generation tool is provided that post-processes the data gathered during a benchmark run and produces a report in accordance with the benchmark specification.

IMS Bench SIPp is based on a modified SIPp and still supports the original SIPp scenario commands as well as a series of extra commands and parameters.

This makes it suitable not only to test IMS core networks or IMS network elements, as targeted by the IMS Performance Benchmark specification, but also standalone SIP proxies, SIP application servers, B2BUAs, etc., whether they are IMS compliant or not. And this can be done while still benefiting from the large-scale benchmarking capabilities, the deep automation, and the report generation functionality of IMS Bench SIPp.

Compared to the original SIPp, it also adds a more realistic traffic profile with its multi-scenario support and the use of a statistical distribution for scenario initiation.

Here are examples of reports generated by IMS Bench SIPp for two hypothetical systems:

- Example Report with a hypothetical IMS Core as SUT
- Example Report with a hypothetical SIP proxy as SUT

A relative drawback of the great flexibility of this test system is that, from the large variety of ways it can be configured and used, only some would be compliant with the ETSI specification. For this reason, it also comes with a configuration tool for generating benchmark configurations that are suitable for benchmark runs according to the specification. It allows the user to enter, through a menu-driven process, the value of parameters as they appear in the specification and translates this to configuration files appropriate for the SIPp-based implementation. Using this script to configure the benchmark is recommended for first time users and for users who want to be sure they run the benchmark in accordance with the specification.

It is however possible to generate custom benchmark configurations for specific cases, outside the scope of the specification (for example testing non-IMS systems). This can be done by editing the configuration and/or scenario files by hand. Even in this case, the configuration tool can still be useful to get an initial version of the configuration and also to learn some aspects of the toolset.

2 Getting IMS Bench SIPp

"IMS Bench SIPp" is released under the GNU GPL license and all the terms of the license apply. It is provided to the SIP and IMS communities by Intel Corporation. It is based on SIPp and we hope the changes made for the implementation of the IMS Performance Benchmark can be useful to the wider SIPp user community as well.

The source tree containing the source code for all the components described here can be obtained from the Subversion repository as described in the installation section of the reference documentation.

Intel does not provide any support nor warranty concerning IMS Bench SIPp. Support can be obtained on a best effort basis through the SIPp mailing lists that Intel engineers also monitor.

3 Tested Platforms

Although SIPp works on most UNIX operating systems, the IMS Bench SIPp has only been tested on Linux Fedora Core 6 and on RedHat Enterprise Linux 4 Update 4. Most other Linux distributions should also work but since performance and timing precision are an important aspect of the test system, one should be very careful and make sufficient validation of the test system when running it on other distributions or other UNIX flavors. Please also note the platform tuning required as explained in the installation section of the reference documentation.

4 Design Objectives

The changes and additions made to SIPp were driven by specific requirements from the ETSI IMS/NGN Performance Benchmark specification, as summarized below (own interpretation):

- A mix of session setup, registration, deregistration and instant messaging scenarios must be executed concurrently
- Scenarios must be selected at random with defined probability of occurrence for each type of scenario (for example 30% messaging, 50% calling, 10% re-registrations, 5% new registrations, 5% de-registrations)
- The number of scenario attempts per time unit must follow a statistical Poisson distribution
- Users involved in the scenario must be selected at random, from the set of users that are suitable for the scenario (e.g. a fresh registration scenario must pick a user who's not registered yet)

As a result of these requirements, the test system must ensure that the scenario execution is precisely known beforehand. This means for example that the test system must be sure that, assuming the System Under Test (SUT) is operating correctly, when it places a call from a user to another one, the call will succeed and follow exactly the expected scenario. This requires that the test system picks a user that is registered. This is not obvious because at the same time users must deregister and later register again because of the mix of scenarios running concurrently. Also there are variations on the calling scenario where the called party rejects the call. In this case, the UAC and UAS sides must have agreed on the expected behavior.

Another objective was to make the SIPp-based test system scalable and highly performing in order to be able to use it for testing very powerful systems and systems with large numbers of users without requiring dozens of test systems attacking one single System Under Test.

Finally, it was also considered important that the test system be capable of generating several times almost exactly the same load over time, despite its intended fundamentally random behavior, so as to help estimate the validity of benchmark scores and to help in troubleshooting test system or SUT issues.

5 Limitations

5.1 Versus the specifications

This implementation of the specification is not complete yet, but provides a sound framework for further development and supports the most important aspects and use cases of the specification. The most challenging technical aspects have already been covered in order to reduce the risk of later encountering a major obstacle that would require significant design changes.

The main limitation is in the set of supported scenarios. The scenarios currently provided lack the "low occurrence" scenarios or variations on already implemented scenarios (Rejected call, Message to not registered user...). Implementing the missing scenarios should not require significant source code changes.

The scenarios have not been thoroughly checked for compliance with 3GPP IMS specifications. The current implementation has everything that was required by the System Under Test used for testing but it should by no means be considered as a reference for 3GPP compliant IMS implementations.

The current implementation does not support IPSec. This means that the connection between the Test Systems and the P-CSCF component of the SUT is using unencrypted SIP messages over UDP or TCP. One should be aware that IPSec processing is likely to have a high performance impact on the P-CSCF component. The current implementation is therefore not suitable for evaluating the performance of a P-CSCF as it would normally be used in 3GPP IMS deployments.

5.2 Other Limitations

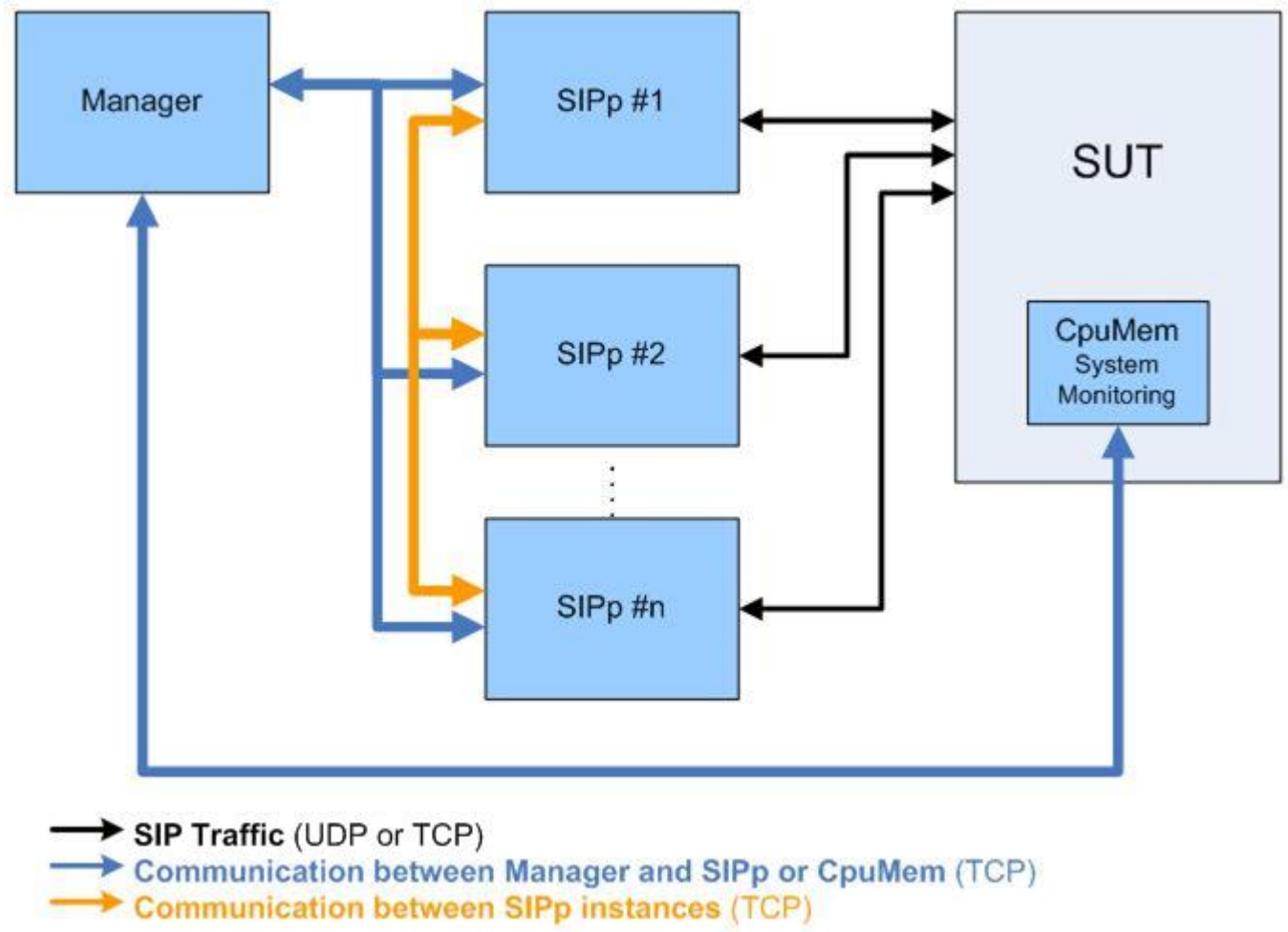
In order to speed up the implementation, some features of the original SIPp have been ignored and are therefore probably not working anymore. They have at least not been tested in this branch. If you wish to use them, you should either not use the IMS Bench SIPp branch or expect some work to do on the source code to fix/restore them.

The following are the known areas that have been intentionally left out:

- Other transports than UDP and TCP (TLS)
- IPv6 support (known to be broken)
- 3PCC operation (presumably broken)
- PCAP Play (not tested)

6 Test System High Level Overview

The Test System consists of one "manager", one or more SIPp load generators, and optionally one or more system monitoring agents for CPU and memory utilization reporting. These are logical components and multiple of them can be running on the same system.



The SIPp load generators execute the various benchmark scenarios. Each SIPp instance is loaded with the full set of scenarios. Originating scenario attempts are started according to a statistical distribution, and the scenario to execute is itself selected at random, as well as the users involved in the scenario. Each SIPp instance has its static set of users that it emulates.

In UDP mode, each user is assigned a unique IP address + UDP port combination. By default, each SIPp instance has a single IP address and assigns different UDP ports to the users, but a compile time option and corresponding configuration allow supporting many IP addresses in a single SIPp instance in order to make the setup more realistic and avoid that traffic be blocked by overflow attack protections at the SUT.

In TCP mode, each SIPp instance has a single IP address and creates one pair of TCP sockets to the SUT. The first socket is used for server side scenarios, and the second one is used for client side scenarios. All users emulated by the SIPp instance share this single pair of TCP sockets.

The manager is responsible for

1. Configuring the SIPp instances: uploading the scenarios, sending configuration data (e.g. parameters that are used in the scenarios, list of other SIPp instances)
2. Carrying out the various steps of the benchmark run by instructing the SIPp instances to set the ratios of each scenario in the scenario mix and the rate of scenario attempts as defined in the benchmark configuration
3. Monitoring the failure rate in order to stop the benchmark when a defined threshold is exceeded (max % of Inadequately Handled Scenarios)
4. Logging system resource utilization (CPU, MEM) reported by the system monitoring agents (from the SUT and/or the Test Systems)

During a benchmark run, the SIPp instances dump measurements (scenario attempts and outcome, timing measurements, retransmissions...) into local files. In addition, they also communicate summary information (number of scenario attempts made, number of failed scenario attempts...) back to the manager so it can monitor the evolution of the benchmark run and stop it in case failure thresholds are exceeded.

During the run, the manager also writes a benchmark raw report that contains all relevant configuration information as well as information about all the steps (changes in scenario mix and/or scenario attempt rate) it performs.

After the run, a post-processing tool loads the benchmark raw report, looks up the IP addresses and path information for the various SIPp instances, fetches the data files that they produced during the run, and generates a report in the form of an HTML file and a series of PNG gnuplot graphs, all packed within an MHTML file (Multipurpose Internet Mail Extension HTML - RFC 2557).

The manager and the SIPp instances communicate over TCP connections. The SIPp instances also communicate together over TCP for the non-SIP messages they exchange (user reservation, timing data).

In order to collect system statistics from the SUT, according to the ETSI benchmark spec, a small monitoring agent (called CpuMem) must be running on the SUT and communicates over TCP with the manager. This agent is included in the source tree and is known to work on Linux and Solaris. This is however optional if report compliance with the spec is not an issue.

7 New Features and Changes to SIPp

Here is a non-exhaustive list of "feature-level" changes that have been made to SIPp. Many of them are probably only really useful in combination with others but they are listed separately for the sake of clarity.

Change	Details
Multiple scenarios support per SIPp instance	Multiple scenarios can be loaded by a single SIPp instance. Each scenario has its own statistics, etc. Keyboard commands can be used to switch between scenarios (to see the corresponding data on screen) and potentially to change the scenario being executed.
Multiple SIPp instances remotely controlled by a central 'manager'	Typical setup includes 1 manager (new piece of code not doing any SIP processing) that coordinates multiple SIPp 'agent' instances. The SIPp instances can be running on the same physical system or on different ones. This should allow nice scaling. The manager feeds the same set of scenarios to each SIPp instance before starting a run and then instructs the SIPp instances to change the scenario attempt rate according to a configuration file. The configuration specifies, for each step of the run, the occurrence rate of each scenario, as well as the rate of scenario attempts, as constant or as increasing steps.
Users and user pools	Each SIPp has a set of users that it represents and whose data it loads from a data file. Users are placed into pools. New scenario commands allow picking a user at random from a specified pool. User pools are used to provide a simple representation of user state. For example a calling scenario picks a user from the pool of already registered users. The registration scenario picks users from the "not registered" pool.
Inter-SIPp instances control messaging (a sort of much extended 3PCC)	If the scenario requires interactions with a server side counterpart (e.g. UAC->SUT->UAS), the client side scenario has a new command to allow the SIPp playing the client role to select a partner SIPp instance at random (from those that registered with the manager). It then sends a custom message (over a separate TCP transport which would normally run on a separate LAN from the network under test) to the partner telling it the server scenario that it requires as well as some extra data so the server side can later identify the first SIP message of the scenario when the client side sends it. The server scenario then typically starts - even before a first SIP message is received - by selecting a user from an appropriate local pool and sending a response to the client side SIPp telling it the user URI that can be used for the scenario (i.e. the To URI). This allows the client side to really start the SIP part of the scenario. This scenario user reservation procedure makes it possible to guarantee the execution of the scenario - assuming the SUT operates correctly - and also allows the individual scenarios to remain very simple as they don't have to accommodate multiple possible paths and outcomes.
One UDP port per User (in UDP mode)	In order to be as realistic as possible, each user is associated with its own IP address + UDP port combination. Default operation uses a single IP address. One can then run multiple SIPp instances on the same system, each running on a different (real or virtual) IP address.
Multiple IP addresses (in UDP mode)	Optionally, to make it even more realistic or to optimize performance of the test system (Linux IP stack may scale better with the number of IP addresses than with the number of ports), it is also possible to distribute the users onto many IP addresses within the same SIPp instance.

Change	Details
One pair of TCP sockets per SIPp instance (in TCP mode)	Since each SIPp instance can simultaneously execute multiple scenarios, possibly with both the client side user and the server side user represented by the same SIPp instance, the incoming SIP traffic must be properly routed to the relevant emulated user. In TCP mode, this is efficiently achieved by means of a single pair of TCP sockets per SIPp instance. The first socket carries SIP traffic for server side scenarios, and the second one is used for client side scenarios. All users represented by the SIPp instance share this single pair of TCP sockets. As in UDP mode, multiple SIPp instances can run on the same system, each one assigned to a different (real or virtual) IP address.
Poisson distribution for scenario attempts	New scenario attempts can be initiated following a statistical Poisson distribution; the user reservation procedure is scheduled so that the actual SIP scenario start follows the Poisson distribution.
Timing measurements between different SIPp instances	New scenario commands were also added in order to allow agent and server side to exchange timing information, and also to allow performing computation on timing measurements. This makes it for example possible to compute the time it took for the INVITE to get from the UAC, through the SUT, to the UAS. Scenario XML files can also specify maximum values for timing measurements (direct or computed from other measurements or timestamps, local or remote). These maximum values are checked at the end of the call and the call is marked as failed in case a maximum is exceeded (even though the scenario might have reached its end without actual timeout or other error). The manager collects the counters of attempted and failed scenarios (due to scenario error or exceeded max time) and determines when a run or step has failed (i.e. percentage of failed scenario attempts went above a threshold - Design Objective in IMS Benchmark spec) and stops the run.
User variables	User variables are similar to call variables but are attached to a user and can therefore be carried over from one scenario attempt to the next. Example: store the Service-Route received during the registration scenario and use it when later placing a call or sending an IM.
Performance improvements	Under Linux, epoll() is now used to reach a much higher timing precision in scheduling new calls and also to significantly lower the CPU utilization of SIPp. The keyboard handling thread was removed and replaced by polling on stdin, and the remote control thread was also integrated into the main polling loop so that each SIPp instance runs as a single threaded process. One can run multiple instances to take advantage of multi-core systems.

Change	Details
Scenario statistics	A new statistics file is created with a line for each scenario attempted, indicating the scenario executed, the start time, the result (success or in case of failure, which case of failure) and the timing measurements as listed in the scenario XML file.
Report Generator	A perl script allows post-processing the data gathered during a run and producing a report (HTML + gnuplot PNG graphs) matching the requirements of the IMS Performance Benchmark spec. It can also be customized in many ways through a report configuration file.
Various	New distribution supported for pause durations: Poisson Added on_unexpected message and scenario attribute in order to jump to failure case label in scenario Possibility to enable/disable default SIP behavior on a per scenario basis "Generic parameters" to be used in scenarios (e.g. %RING_TIME) can be provided in manager configuration and loaded to SIPp instances (in addition to being supported on SIPp command line).