



**HELSINKI METROPOLIA UNIVERSITY OF APPLIED SCIENCES**

**Information Technology**

**Multimedia Communications**

**MASTER'S THESIS**

**VIRTUALIZATION-BASED FAULT TOLERANCE**

**Author: Jagan Mohan Devinani**

**Supervisor: Kari Björn, Development Manager**

**Instructor: Michael Zhidovinov, Senior Specialist**

**Approved: \_\_\_\_ \_\_\_\_ 2011**

**Kari Björn, Development Manager**

## **PREFACE**

This has been a great learning process for me to work with the topic related to virtualization. This work has been extremely demanding and required a sustained effort for more than one year. Especially, carrying out the experiments with the most advanced hardware at the laboratory in the Nokia Siemens Networks has been an invaluable learning experience.

I would like to thank my supervisor Kari Björn, from Metropolia University of Applied Sciences, for his support and valuable comments during the writing of this thesis, as well as Jonita Martelius for the help provided with the English language checking of this study.

I would like to thank my instructor Michael Zhidovinov and all other colleagues at Nokia Siemens Networks, for their invaluable support and guidance.

Finally, I would like to thank all the members of my family and friends for their relentless support and encouragement.

May 26, 2011  
Espoo, Finland

Jagan Mohan Devinani

**HELSINKI METROPOLIA UNIVERSITY OF APPLIED SCIENCES**

**ABSTRACT OF THE MASTER'S THESIS**

Name: Jagan Mohan Devinani	
Title: Virtualization-Based Fault Tolerance	
Date: May 26, 2011	Number of pages: 49
Degree Programme: Information Technology	Specialization: Multimedia Communications
Supervisor: Kari Björn, Development Manager	
Instructor: Michael Zhidovinov, Senior Specialist	
<p>This thesis presents an evaluation of the Virtualization-Based Fault Tolerance solutions and approaches available from different vendors.</p> <p>The performance of a SIP (Session Initiation Protocol) based application among “native environment”, “virtual environment” and “virtual environment with fault-tolerance feature turned on” are compared.</p> <p>The experiment was carried out with “VMWare ESXi” as the virtualization layer and “VMWare vSphere” as the management client to administer the “VMWare ESXi” platform.</p> <p>The experiment was carried out in a client-server model with UAC (User Agent Client) acting as the SIP client and SUT (System Under Test) as the SIP server. The “IMS-Bench SIPp” software was used in UAC and the “opensips” software was used in the SUT.</p> <p>The results obtained from the experiments carried out in the three different environments are compared with respect to the SAPS (Scenario Attempts Per Seconds). The SAPS in “native environment” are one hundred and thirty nine, in “virtual environment” are one hundred and in “virtual environment with fault-tolerance feature turned on” are thirty four.</p>	
Key words: Virtualization, Fault Tolerance, High Availability, Hypervisor	

# TABLE OF CONTENTS

PREFACE

ABSTRACT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

ABBREVIATIONS

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	Research Methods .....	2
1.2	Study Structure .....	3
<b>2</b>	<b>BACKGROUND</b>	<b>4</b>
2.1	Virtualization Concepts .....	4
2.2	Virtualization-Based Fault Tolerance Techniques .....	8
2.3	Virtualization-Based Fault Tolerance Solutions.....	9
2.4	Initial Prototype of Hypervisor .....	10
2.5	VMWare vSphere Fault Tolerance.....	11
2.5.1	<i>VMware Fault-Tolerance Functionality</i> .....	11
2.5.2	<i>VMWare Fault-Tolerance Architecture</i> .....	12
<b>3</b>	<b>DESCRIPTION OF THE EXPERIMENT</b>	<b>15</b>
3.1	Native Environment .....	19
3.1.1	<i>Configuring and Starting SUT</i> .....	20
3.1.2	<i>Configuring and Starting Manager</i> .....	21
3.1.3	<i>Configuring and Starting UAC</i> .....	23
3.1.4	<i>Report Generation at Manager</i> .....	24
3.2	Virtual Environment .....	24
3.2.1	<i>Configuring and Starting SUT</i> .....	27
3.2.2	<i>Configuring and Starting Manager</i> .....	27
3.2.3	<i>Configuring and Starting UAC</i> .....	28
3.2.4	<i>Report Generation at Manager</i> .....	29

3.3	Virtual Environment with FT Switched ON .....	30
4	RESULTS AND ANALYSIS	32
4.1	Performance in Native Environment.....	32
4.2	Performance in Virtual Environment .....	34
4.3	Performance in Virtual Environment with FT Switched ON .....	36
4.4	Performance Improvement Techniques in Virtual Environment.....	45
5	SUMMARY	47
6	REFERENCES	48

## LIST OF FIGURES

Figure 2-1 Before Virtualization.....	5
Figure 2-2 After Virtualization .....	5
Figure 2-3 High Availability .....	7
Figure 2-4 Fault Tolerance.....	7
Figure 2-5 Transparent Failover .....	8
Figure 2-6 Hypervisor based Fault Tolerance .....	10
Figure 2-7 VMWare vSphere Fault Tolerance [5:1].....	12
Figure 2-8 High level Architecture of VMWare Fault Tolerance [6:4] .....	13
Figure 3-1 Physical host in Native Environment.....	16
Figure 3-2 Physical host in Virtual Environment.....	17
Figure 3-3 Experiment setup in Native Environment.....	19
Figure 3-4 Starting opensips on SUT on IP address 10.20.106.105 .....	21
Figure 3-5 Starting Manager on static IP address 10.20.106.106.....	22
Figure 3-6 Starting the first UAC on static IP address 10.20.106.107 .....	23
Figure 3-7 Experiment setup in Virtual Environment.....	25
Figure 3-8 Starting opensips on SUT on VM_4_SUT_105.....	27
Figure 3-9 Starting Manager on VM_1_Manager_106.....	28
Figure 3-10 Starting UAC on VM_2_UAC_107 .....	29
Figure 3-11 VMWare-vSphere Cluster view with FT switched on.....	30
Figure 4-1 Configuring all the Network Interface Cards.....	46

## LIST OF GRAPHS

Graph 4-1 CPU utilization of SUT in Native Environment.....	34
Graph 4-2 CPU utilization of SUT in Virtual Environment.....	36
Graph 4-3 CPU utilization of SUT in Virtual Environment with FT switched on .	38
Graph 4-4 Inadequately Handled Scenarios.....	39
Graph 4-5 Inadequately Handled Scenarios for ims_uac scenario.....	40
Graph 4-6 Retransmissions for All Scenarios.....	41
Graph 4-7 Retransmissions for ims_uac scenario .....	42
Graph 4-8 Retransmissions for ims_reg scenario.....	43
Graph 4-9 Calling Use Case for Session Setup .....	44
Graph 4-10 Calling Use Case : Delay between BYE and OK.....	45

## LIST OF TABLES

Table 4-1 Performance in three different environments.....	32
Table 4-2 Resource utilization of SUT in Native Environment.....	33
Table 4-3 Resource utilization of SUT in Virtual Environment.....	35
Table 4-4 Resource utilization of SUT in Virtual Environment with FT switched on .....	37

## ACRONYMS / ABBREVIATIONS

HW	Hardware
SW	Software
CPU	Central Processing Unit
OS	Operating System
RHEL	RedHat Enterprise Linux
VMWare ESXi	Virtualization Platform from VMWare
VMWare vSphere	Management SW to administer the ESXi platform
Xen	Virtualization Platform from Xen open source community
FT	Fault Tolerance
VFT	Virtualization Based Fault Tolerance
HA	High Availability
VM	Virtual Machine
VMM	Virtual Machine Monitor
CA	Continuous Availability
FDRT	Fined-grained Dirty Region Tracking
SMP	Symmetric Multi Processing
NIC	Network Interface Card
SIP	Session Initiation Protocol
IP	Internet Protocol
SAPS	Scenario Attempts Per Seconds
IHS	Inadequately Handled Scenarios
TS	Test System
UAC	User Agent Client/Load Generator
SUT Manager	System Under Test Test harness system to control the tests, collect the data from SUT and to generate report.
opensips	Open SIP Server



IMS-Bench SIPp	The SIP Proxy for generating SIP traffic
{CPB}	Directory of Control Plane Benchmark
{opensips}	Installation directory of opensips-1.6.2-notls at SUT
{IMS_Bench@SUT}	Installation directory of IMS Bench SIPp at SUT
{IMS_Bench@MANAGER}	Installation directory of IMS Bench SIPp at the Manager
{IMS_Bench@UAC_x}	Installation directory of IMS Bench SIPp at the UAC
BYE	Message to initiate a SIP call termination
OK	Acknowledgement to BYE message during SIP call termination

# 1 INTRODUCTION

This thesis presents an evaluation of the Virtualization-Based Fault Tolerance solutions and approaches available from different vendors. This thesis compares the performance of a SIP (Session Initiation Protocol) based application among “native environment”, “virtual environment” and “virtual environment with FT (Fault-tolerance) feature turned on”. The experiment was carried out with “VMWare ESXi” as the virtualization layer and “VMWare vSphere” as the management client for administering the “VMWare ESXi” platform. The experiment was carried out in a client-server model with UAC (User Agent Client) acting as the client and SUT (System Under Test) as the server. The UAC (running “IMS-Bench SIPp” software) generates the SIP traffic to load the SUT (running “opensips” software). Three physical hosts represent the Manager, SUT and UAC respectively. The Manager was used as a test harness system to control the execution of the tests, to collect the data from SUT and to generate the report based on the collected data. SUT was the target system, which gets loaded with SIP (Session Initiation Protocol) traffic. The SUT was subjected to the continuously increasing SIP traffic. The test gets stopped automatically as soon as the resources of the SUT were exhausted. The UAC was acting as the SIP proxy for generating SIP traffic, which used to load the SUT.

## Motivation

There are no experiments carried out in this space to compare the performance of an application in the “native environment”, “virtual environment” and “virtual environment with FT feature turned on”. The application utilized for carrying out the experiments is “IMS-Bench”, which is used as a standard SIP application across the telecom space by many organizations.

## Research Problem

The aim of the present study was to evaluate the “Virtualization-Based Fault Tolerance” solutions and approaches available from different vendors. As well as to compare the

performance of a SIP(Session Initiation Protocol) based application among “native environment”, “virtual environment” and “virtual environment with fault-tolerance feature turned on”.

## **Scope**

The experiment was carried out with “VMWare ESXi” platform only. This platform serves as the infrastructure for providing the virtualization.

There is another competent platform namely Xen [12], which is an open source product providing the virtualization infrastructure with “fault tolerance” feature. But the experiment was not carried out using this platform as this platform lacked the “fault tolerance” feature at the time, when this experiment was carried out.

## **1.1 Research Methods**

The following research methods were used to carry out this study.

Desk research was used to research the background information related to virtualization, VFT implementation techniques and to study about the products VMWare ESXi, Xen.

Experiments in the controlled environment were carried out with VMWare ESXi platform providing the virtualization layer and VMWare vSphere as the management SW to administer the ESXi platform. The experiment has been also carried out with another open source product namely Xen to some extent. But it was realized that the VFT feature in Xen was not matured enough to proceed further with the experiments.

Interviews were conducted with the colleagues at the Nokia Siemens Networks namely Machael Zhidovinov and Kari Hautio. The interviews with Machael Zhidovinov helped in scoping the thesis work, downgrading the system configuration in order to reach the 100% CPU utilization, removing the irrelevant data (after the experiment) to generate accurate reports, optimizations in virtual environment. The interviews with Kari Hautio helped to configure the Storage Area Network to store the Virtual Machines.

## 1.2 Study Structure

This study consists of five chapters.

Chapter two presents the background information related to the concepts of virtualization, VFT (Virtualization-Based Fault Tolerance) techniques and solutions. The products, which implemented these VFT techniques. Also presents the initial prototype of Hypervisor implementation. Also discusses the VMWare vSphere VFT functionality and the architecture. Chapter three presents the procedure to carry out the experiment, required HW(Hardware) and SW(Software), the experimental setup in “Native Environment”, “Virtual Environment” and “Virtual Environment with Fault-Tolerance switched on”. Configuring and starting the SUT, Manager, UAC and the report generation at the Manager.

Chapter four presents the results and analysis with respect to the performance in “Native Environment”, “Virtual Environment” and “Virtual Environment with FT switched on”. The analysis of failover situation with respect to IHS (Inadequately Handled Scenarios), delays and retransmissions with respect to the number of SAPS (Scenario Attempts Per Seconds) are discussed. The calling use case with respect to delays in session setup time and delays during session termination time are analysed. Also presents the performance improvement techniques in Virtual Environment. Chapter five presents the summary of this study. Also includes the discussion and conclusions about the most important findings, evaluates the key issues and points to potential future work in the area of Virtualization-Based Fault Tolerance.

## 2 BACKGROUND

This chapter first introduces virtualization as a general concept, then presents the VFT (Virtualization-based Fault Tolerance) techniques and VFT solutions. Then presents the initial work to build a prototype of the hypervisor to support virtualization. Finally, the architecture of VMWare vSphere Fault-tolerance is explained.

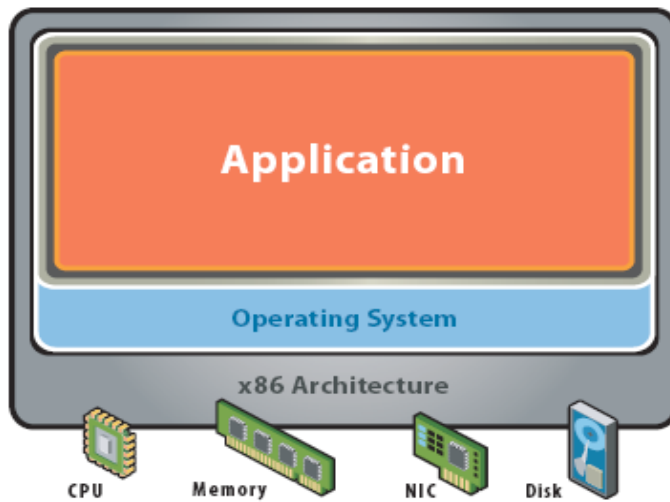
The research in the VFT (Virtualization-based Fault Tolerance) domain had started way back in 1990's. The initial work to build a prototype of the hypervisor to support virtualization and to further demonstrate the virtualization-based fault tolerance was built in 1996 [1:14]. The detailed description about the Initial Prototype of Hypervisor is furnished in the Chapter 2.3. The interest towards virtualization has increased as it offers the efficient utilization of the resources. The latest HW (Hardware) from various vendors supports virtualization at the HW level.

### 2.1 Virtualization Concepts

This chapter introduces the general concepts and terminology associated with virtualization such as "Virtual Machine", "Virtual Machine Monitor (Hypervisor)", "High Availability", "Fault-Tolerance", "Virtualization-Based Fault Tolerance (VFT) " and "Transparent Failover".

#### Virtualization

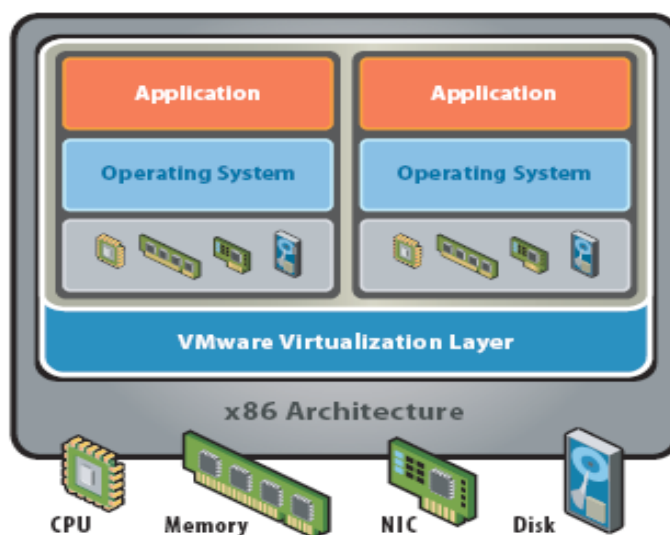
The term virtualization broadly describes the separation of a resource or request for a service from the underlying physical delivery of that service. The virtualization technologies provide a layer of abstraction between computing, storage and networking hardware, and the applications running on it. Figure 2-1 depicts the environment before virtualization with traditional infrastructure containing and HW (hardware), OS (Operating System), SW (software) and applications.



**Figure 2-1 Before Virtualization**

It has a single OS (Operating System) image per machine. In this environment, the SW and HW are tightly coupled. The resources are usually underutilized, inflexible and the infrastructure is expensive. It often creates a conflicting situation, when the multiple applications are executed on the same machine.

Figure 2-2 depicts the environment after virtualization with virtualization layer introduced on top of the HW. The virtualization layer helps to create multiple VMs (Virtual Machine) and each VM mimics a physical host.



**Figure 2-2 After Virtualization**

It is possible to manage OS and application as a single unit by encapsulating them into VM. Each VM can be installed with OS (Operating System) and applications on top of

the OS. In this environment, the OS and applications are independent of underlying HW. The resources are efficiently utilized.

### **Virtual Machine (VM)**

The VM is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine. The VMs allow the sharing of the underlying physical machine resources between different VMs, each running its own operating system. The advantages of VMs are that multiple OS environments can co-exist on the same computer, in strong isolation from each other. The application provisioning, maintenance are easy and the services such as high availability and disaster recovery are provided as part of the virtual infrastructure. The disadvantage of VM is that it is less efficient than a real machine because it accesses the hardware indirectly.

### **Virtual Machine Monitor (VMM) or Hypervisor**

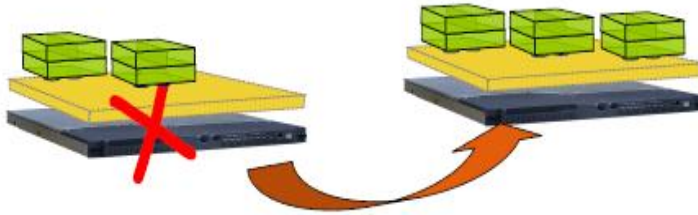
The VMM is the software layer providing the virtualization. A hypervisor can run on bare hardware (Type 1 or native VM) or on top of an OS (Type 2 or hosted VM).

### **Virtualization-Based Fault Tolerance (VFT)**

The Fault-tolerance systems designed based on the virtual infrastructure, leveraging on the capabilities of virtualization.

### **High Availability (HA)**

The VMs on a failed host are automatically restarted on another available server to maintain the availability of service. The VM is unavailable during this switch over period. Figure 2-3 shows the restart of VM on secondary host, when the primary host running the VM fails.

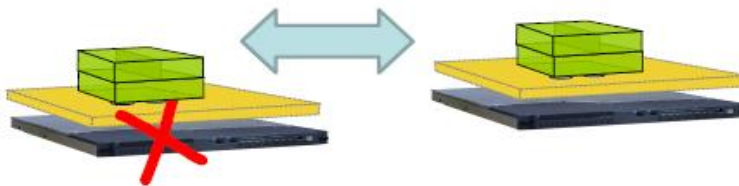


**Figure 2-3 High Availability**

To provide HA in virtual environments, no modifications are needed to the applications and OS. The HA monitors hosts and VMs to keep applications running. The HA reduces downtime but does not eliminate it.

### **Fault-Tolerance (FT)**

The FT is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components.



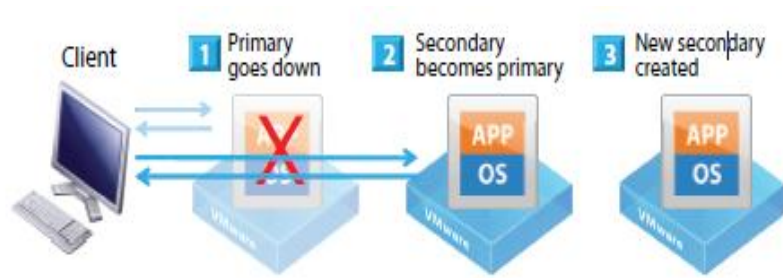
**Figure 2-4 Fault Tolerance**

The VM state is replicated on the Secondary host. Figure 2-4 shows that the VM on secondary host takes over, when the primary host running the VM fails. The downtime is almost zero.

### **Transparent Failover**

Figure 2-5 shows an example scenario depicting the transparent failover situation with respect to VMWare vSphere [16:4]. When the “Physical Machine” running the “Primary VM” fails, the hypervisor on the “Secondary Physical Machine” notices the failure immediately and has full information on pending I/O (Input/Output) operations from the failed “Primary VM” ( it commits all pending I/O ). The hypervisor performs a “go live” operation and becomes the “New Primary”.





**Figure 2-5 Transparent Failover**

The previous dependencies on the old (and now failed) Primary are terminated. The HA ( after the initial failover ) selects a “New Host” and starts a “New Secondary VM Instance” on that host. The entire failover process is automatic and there is no disruption of service, no loss of state.

## 2.2 Virtualization-Based Fault Tolerance Techniques

The following are the common VFT techniques employed to synchronize the state of VM's across Physical Machines.

### Lockstep

Some of the products implementing this technique are Kemari [2] and VMWare-vSphere [5]. The core logic of this technique is that external events are logged on Primary VM and replayed on Secondary VM. Lockstep has efficient synchronization mechanism. The drawback of the lockstep is that the implementation complexity is high as it needs to handle differences between processor families.

Lockstep systems are redundant computing systems that run the same set of operations at the same time in parallel. The output from lockstep operations can be compared to determine if there has been a fault. To run in lockstep, each system is set up to progress from one well-defined state to the next well-defined state. When a new set of inputs reaches the system, it processes them, generates new outputs and updates its state. This set of changes (new inputs, new outputs, new state) is considered to define that step, and must be treated as an atomic transaction; in other words, either all of it happens, or none of it happens, but not something in between.

## **Checkpointing**

Some of the products implementing this technique are Kemari [2] and Remus [3]. Kemari takes advantage of the best of lockstep and checkpointing techniques. The core logic of this technique is that the state of Primary VM is transferred to the Secondary VM frequently. The states of healthy machines are preserved either locally or remotely. In case of failures, the system is rolled back to the most recent checkpoint. The checkpointing involves less domain-specific knowledge of either HW (Hardware) or applications. Hence the complexity of implementation is less. The drawback of the lockstep is that the outputs to devices are delayed for a specific period of time because they are buffered to keep the state of the VMs and the attached devices consistent). It is possible to perform the state checkpointing on the basis of per-process and also on the whole-machine.

## **2.3 Virtualization-Based Fault Tolerance Solutions**

The most popular VFT solutions and the logical aspects of these approaches are discussed in brief in this section.

### **VMWare vSphere**

VMWare vSphere [5:1] eliminates the downtime even in the event of a complete host failure. It is based on vLockstep technology and it works by keeping primary and secondary VMs in synchronization across different ESX Hosts. The synchronization is maintained by recording the input executed on primary and replaying the same on backup. The replaying on the backup is carried out via FT logging. If the host running an FT enabled VM fails then the VM will continue running on secondary host seamlessly. There is no loss of system state. The control and presentation of the VM will be transferred live to the backup. The detailed description about VMWare vSphere is furnished in the chapter VMWare vSphere Fault Tolerance.

## Kemari

Kemari [4:3] takes advantage of both checkpointing and lockstep. Kemari checkpoints primary instance, when VMM is going to duplicate external events to the backup instance. The buffering of external events can be avoided. The detailed description of Kemari is discussed further in APPENDIX F.

## Remus

Remus [3] works by live-migrating VM from the primary physical host to the backup physical host continuously. For live-migrating the VM, the guest's disk must be available through shared storage both on primary and backup. Remus copies over only the dirtied memory states to reduce amount of memory states to be transferred between primary and backup during checkpointing. The checkpointing frequency can be as high as every 25 msec. Remus buffers the external events until the end of each synchronization between primary and the backup VM. The detailed description of Remus is discussed further in APPENDIX G.

## 2.4 Initial Prototype of Hypervisor

The initial work to build a prototype of the hypervisor to support virtualization and to further demonstrate the virtualization based fault tolerance was built in 1996 [1:14]. The setup is as shown in Figure 2-6. [1:14]

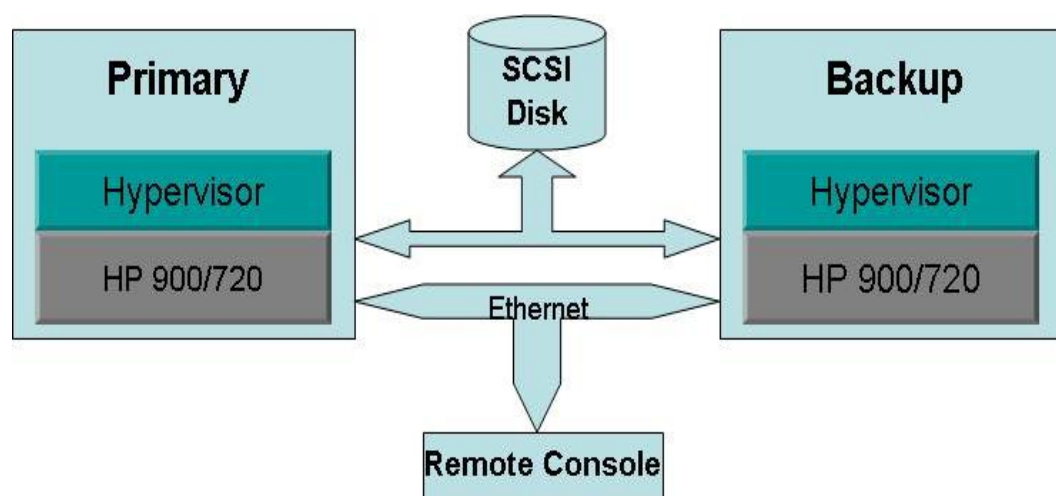


Figure 2-6 Hypervisor based Fault Tolerance

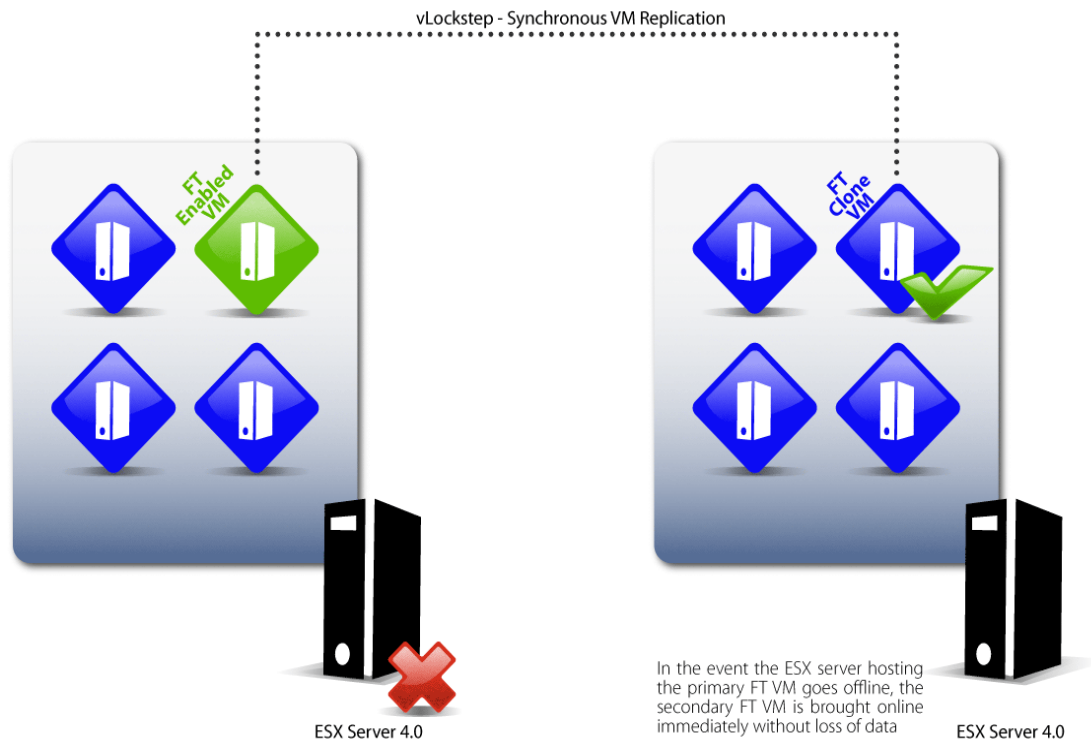
The initial prototype SW of hypervisor was running on two physical hosts. The hardware on the physical hosts was namely HP 900/720 (the HW vendor was Hewlett-Packard). The hypervisor functions by logging each instruction-level operation to primary machine and then replaying the logged operations on top of the backup machine. The advantages of this approach are due to the reason that the logging is done in VMM (Virtual Machine Monitor) or Hypervisor. This logging of instructions in the VMM is possible due to virtualization. Also no additional modifications are needed to the OS(Operating System) or applications. The drawbacks of this approach are as follows. The domain-specific knowledge is required to log events, The “deterministic replay of logging events” relies on the target architecture, The deterministic replay requires an implementation on a specific VMM. The deterministic replay depends on the order in which multiple cores access shared memory for multi-core CPUs.

## **2.5 VMWare vSphere Fault Tolerance**

This chapter describes the functionality and the architecture of the VMWare FT (Fault-Tolerance).

### **2.5.1 VMware Fault-Tolerance Functionality**

The VMWare FT creates a clone of a running “FT Enabled VM” on another Host within the VMWare cluster as shown in Figure 2-7. The changes made on the “FT Enabled VM” are replicated to the “FT Clone VM” in real time. Both VMs have the same IP address and the same MAC address.



**Figure 2-7 VMWare vSphere Fault Tolerance [5:1]**

“The ESX Hosts send continuous heartbeat signals between the primary and secondary hosts to detect any failures. If the heartbeat is lost then the secondary ESX hosts will immediately know about it and the virtual machine will failover without any loss of data. The application continues to function on the secondary server without interruption.” [5:2]

## 2.5.2 VMWare Fault-Tolerance Architecture

The VMware FT is based on the vLockstep technology. This chapter describes some of the key aspects of VMware vLockstep technology, i.e. the “Deterministic Record/Replay” and “Fault Tolerance Logging Traffic”.

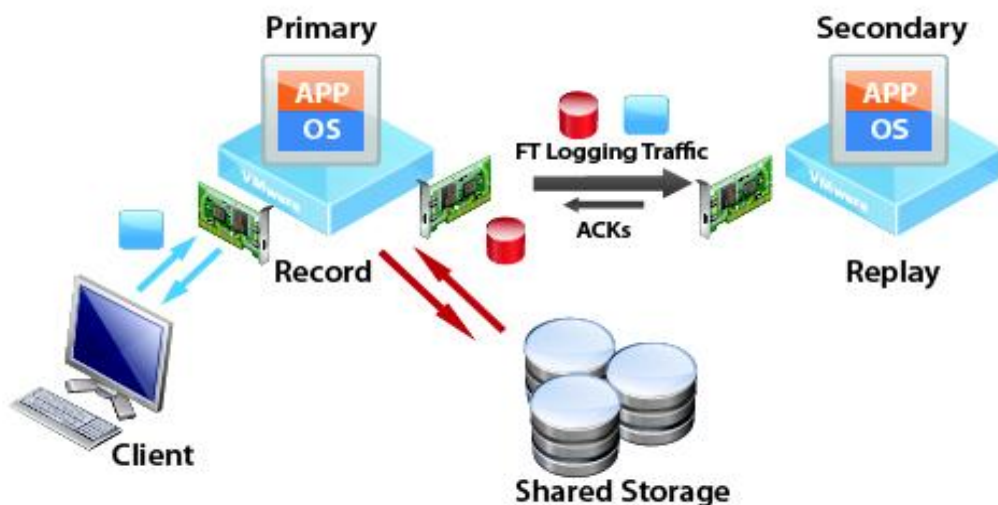
### Deterministic Record/Replay

Deterministic Record/Replay [6:3] is a technology that allows to capture the execution of a running VM for later replay. Deterministic Replay of computer execution is challenging since external inputs such as incoming network packets, mouse, keyboard, disk I/O completion events operate asynchronously and trigger interrupts that alter the

code execution path. Deterministic Replay can be achieved by *recording non-deterministic inputs* and then by injecting those inputs at same execution point during replay. This method greatly reduces processing resources and space as compared to exhaustively recording and replaying individual instructions.

## Fault Tolerance Logging Traffic

The high level architecture of VMware Fault Tolerance is shown in Figure 2-8.



**Figure 2-8 High level Architecture of VMWare Fault Tolerance [6:4]**

The Primary and Secondary VMs (Virtual Machines) share the same virtual disk on shared storage, but all I/O operations are performed only on the Primary host. When the FT is enabled for a “Primary VM” [6:3], the “Secondary VM” gets created by live-migrating the memory contents of the “Primary VM” using VMware vMotion. Once the “Secondary VM” is live, it runs in lockstep and mirrors the guest instruction execution of the “Primary VM”. The Hypervisor running on the Primary Host [6:3] captures the external inputs to the VM and transfers them asynchronously to the secondary host. The Hypervisor running on the Secondary Host [6:3] receives these inputs and injects them into the replaying VM at the appropriate execution point.

“The communication channel between the Primary and the Secondary host is established by the hypervisor using a standard TCP/IP socket connection and the traffic flowing between them is called FT logging traffic. By default, incoming network traffic and disk reads at the primary virtual machine are captured and sent to the

secondary, but it is also possible to make the secondary virtual machine read disk I/O directly from the disk.” [6:4].

This chapter has now explained the general concepts of virtualization, VFT techniques and VFT solutions, the initial prototype of the hypervisor to support virtualization, the architecture of VMWare vSphere Fault-Tolerance.

### **3 DESCRIPTION OF THE EXPERIMENT**

“The benchmark system simulates a group of SIP end point users who perform SIP operations through the SUT (System Under Test).” [13:2.1]. The SUT is the physical host running the opensips (OpenSIP Server). The UAC (User Agent Client) is another physical host running the “IMS-Bench SIPp” [11], which generates the SIP traffic. The Manager is the test harness system, which is responsible for coordinating the test. The Manager carries out the tasks of starting and stopping the TS (client workload generator) on UAC, collecting and aggregating the results from the SUT, determining if a run was successful and generating the reports based on the results obtained from SUT.

The experiment was carried out in all the three different environments until the resources of system such as CPU utilization reaches 100%. The following scenarios are extensively measured and observed during the experiment as those are the most important results obtained through this experiment. The remaining measurements obtained through this experiment are not extensively discussed to limit the scope of this thesis.

#### **1. Performance (CPU utilization)**

The performance means the number of SAPS (Attempts Per Seconds) achieved when the CPU utilization of the physical host representing the SUT reaches 100% ( which means that all the CPU resources of SUT are exhausted ). The number of SAPS executed in each iteration is increased gradually until the resources of the SUT are exhausted. The SAPS are defined in XML (Extensible Markup Language) configuration files. The examples of SAPS are such as SIP subscribers sending registration requests to SUT, establishing call between different SIP subscribers.

#### **2. Delays**

The delays in transmission of the SAPS occur, when the failover situation occurs in the virtual environment. The consequences of failover situation



appear as delays in transmission of the data. It is obvious that the effect of failover is more, when the delays in transmission of the SAPS increases.

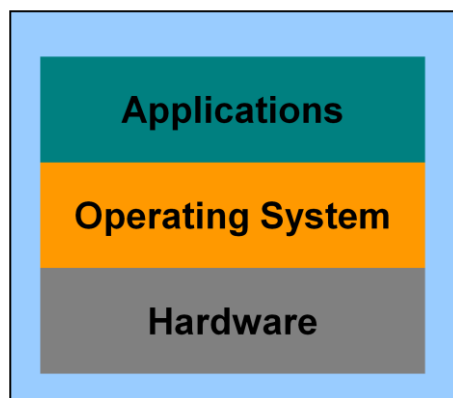
### 3. Retransmits

The retransmission of the SAPS happens, when the failover situation occurs in the virtual environment. It is obvious that the effect of failover is more, when the retransmission of the SAPS increases.

The experiment was carried out in the following three different environments namely "Native Environment", "Virtual Environment" and "Virtual Environment with FT feature turned on".

#### Native Environment

Figure 3-1 shows the logical architecture of how the components of a physical host fit together in the native environment. The native environment contains the HW, OS and applications deployed on top of the OS. There is no virtualization layer in the native environment. The instructions being processed by the applications are executed at the HW level by utilizing the OS.



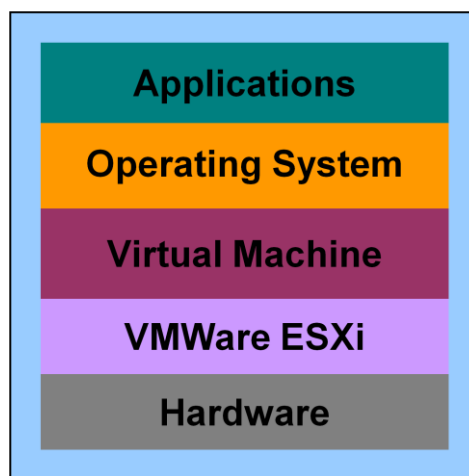
**Figure 3-1 Physical host in Native Environment**

Only a single OS image can be installed per physical machine. The SW and HW are tightly coupled. It can often lead to conflicts if multiple applications are run on the same

machine. The resources are underutilized. The infrastructure can be costly and also inflexible.

## Virtual Environment

Figure 3-2 shows the logical architecture of how the components of a physical host fit together in the virtual environment. The physical host in the virtual environment contains the HW, virtualization layer, virtual machine installed with OS and applications deployed on top of the OS.



**Figure 3-2 Physical host in Virtual Environment**

In the virtual environment, it is possible to create multiple VMs(virtual machine) on a single physical host by utilizing the virtualization layer. Each VM can be installed with a separate OS. The OS and applications can be managed as a single unit by encapsulating them into VM. The applications and OS are independent of HW. The resources are efficiently utilized in the virtual environment.

## Virtual Environment with FT feature turned on

The logical architecture of how the components of a physical host fit together in the “Virtual Environment with FT switched ON” is as shown in Figure 3-2. This is exactly same as described in the “virtual environment”. The only difference is that the Fault Tolerance feature is turned on in the cluster environment.

## Hardware

This experiment was carried out with three physical hosts. Each of the physical hosts had the “Intel Xeon X5460” processor [14]. This processor supports the virtualization at HW level, which is important to make the Fault Tolerance feature to work fine. The virtualization support at the HW level is available in most of the latest processors supplied by the HW vendors. The detailed specification of the HW used in this experiment is shown in APPENDIX A.

## Software

The following software was used in the experiment. All the software components used in this experiment are free except “VMWare ESXi” and “VMWare vSphere”. The evaluation versions of “VMWare ESXi” and “VMWare vSphere” were used in this experiment and were promptly uninstalled from the physical hosts before the expiration of the evaluation license.

- **Operating System**

The operating system used in this experiment was “RedHat Enterprise Linux 5.5 (RHEL5.5)”. This experiment can be carried out with any other variant of Linux distributions available from other vendors and open source software communities.

- **MySQL [15]**

The “MySQL” database software was used in this experiment to store the SIP subscriber information. This database is running on the physical host representing the SUT. The SUT authenticates the SIP subscribers by verifying the existing information of the subscribers available in the database.

- **opensips [10]**

This software acts as the SIP server. The version used in this experiment was “opensips-1.6.2-notls”

- **IMS-Bench SIPp [11]**

This software is used to generate the SIP traffic, which acts as the SIP client.

- **Control Plane [9]**

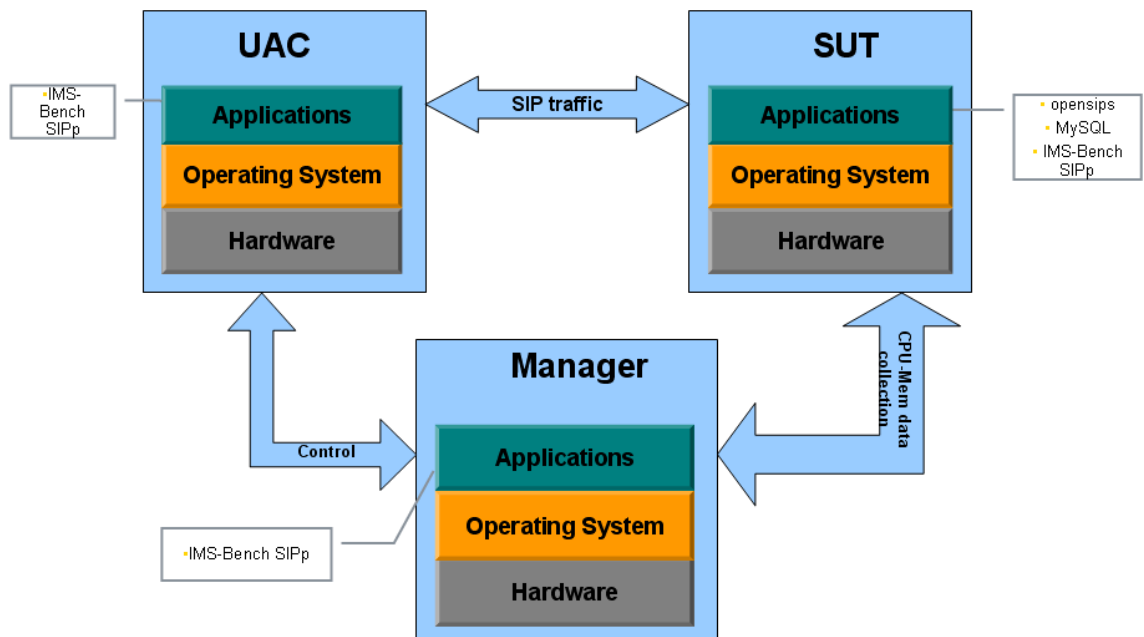
This software provided the configuration data, which was used as a reference configuration while configuring the experiment for the first time.

### Prerequisite

The SW that need to be installed and the configurations that are done prior to starting the experiment are described in APPENDIX D. These are the mandatory steps that need to be performed before installing any other SW and carrying out any other configuration changes on SUT, UAC and Manager.

## 3.1 Native Environment

Figure 3-3 shows the logical architecture of how the components of the benchmark system fit together.



**Figure 3-3 Experiment setup in Native Environment**

The “Native Environment” contains the HW (Intel Xeon X5460), OS (RHEL-5.5) and applications deployed on top of the OS. Three physical hosts (UAC, SUT, Manager) were needed to carry out the experiments in “Native Environment”.

## **Execution of the Experiment**

The following set of instructions listed below to carry out the execution of the tests are the same in all the three types of environments (Native Environment, Virtual Environment, Virtual Environment with FT switched on). The “ntpd”, “ptpd” services should be started on SUT, Manager and UAC with the commands “./etc/init.d/ntpd start” and “./ptpd/trunk/src/ptpd” respectively.

### **Configure Virtual IPs [11] on UAC**

It is required to run each of the UAC with distinct IP address and port combination can support up to 10000 SIP subscribers. To create a traffic of 100000 subscribers, 10 UACs are needed. Since it is not feasible to have 10 physical hosts, one physical host used as primary UAC and 9 virtual IP addresses are created on top of this physical host. So that 10 UACs can be run simultaneously on a single physical host by configuring virtual IP addresses. After the virtual IPs are created and configured [11] on UAC, the routes are added on both SUT and Manager to all the virtual IP addresses of UAC.

The scope of these virtual IP addresses is confined to UAC only. These virtual IP addresses are not published to the entire network. They do not carry any meaning, when referenced external to the UAC. Hence, when these virtual IP addresses are referenced on SUT and Manager, the routes should be added both on SUT and Manager so that SUT and Manager can route the traffic destined for these virtual IP addresses to the physical IP address of UAC. In turn, UAC can internally route the traffic to one of the virtual IP addresses internally, where the traffic is actually destined to. The routes to the virtual IP addresses (10.20.106.2 to 10.20.106.9) can be added both on SUT and Manager using the command “route add -net 10.20.106.2 netmask 255.255.255.255 gw 10.20.106.107”.

#### **3.1.1 Configuring and Starting SUT**

The SUT is already configured with the static IP address “10.20.106.105”. The command to start the SIP server on the SUT is shown in Figure 3-4.

```

root@localhost:~/opensips-1.6.2-notifs/OPENSIPS/sbin
Window Menu w Terminal Tabs Help
ps/opensips.cfg
root 3360 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3361 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3362 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3363 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3369 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3371 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3372 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3373 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3374 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3379 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3381 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3383 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3385 3345 0 12:42 ? 00:00:00 ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg
ps/opensips.cfg
root 3389 3173 0 12:42 pts/1 00:00:00 grep opensips
[root@localhost sbin]# ./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg

```

**Figure 3-4 Starting opensips on SUT on IP address 10.20.106.105**

The SUT was installed with the MySQL database. The “opensips” SW contains the required scripts to populate the MySQL database with the information of SIP subscribers. The detailed instructions to install the MySQL database and the installation of other software components such as “opensips” are described in detail in the APPENDIX B. The detailed instructions to populate the MySQL database with SIP subscriber information are described in detail in the APPENDIX B. The detailed instructions to start the SIP server (opensips) are described in detail in the APPENDIX B.

### 3.1.2 Configuring and Starting Manager

The Manager will wait until all the UACs alias TSs (Test Systems) are started on the physical host dedicated for UAC. Each of the started UACs gets registered with the Manager. At the Manager side, the key “e” on the keyboard should be pressed to start the execution of the experiment. The experiment stops at the Manager side, when the IHS (Inadequately Handled Scenarios) exceeds 10%. The “CPS” (Calls Per Second) in the “manager.xml” is increased by 10 after each iteration automatically until IHS

reaches 10% during Benchmark run phase. The Manager was already configured with the static IP address “10.20.106.106”. The command to start the Manager is as shown in Figure 3-5.

```

Applications Places System 1:17 AM
Manager 106
File Edit View Terminal Tabs Help
Manager 106 x 105 SUT CpuMem x 106 x root@localhost: ~ x
01:12:15.666| ims_reg 01 (S+F)= 874 F= 468 IHS= 53.54691%
01:12:15.666| ims_uac 02 (S+F)= 17318 F= 7779 IHS= 44.91858%
01:12:15.666| ims_dereg 04 (S+F)= 849 F= 453 IHS= 53.35689%
01:12:15.666| ims_msgc 05 (S+F)= 10195 F= 3023 IHS= 29.65179%
01:12:15.666| *IHS ALL* (S+F)= 34488 F= 14860 IHS= 43.08745%
01:12:15.987| CPU0 7066746ms: 100.0000 MT: 5072736 MF: 3776448
01:12:16.987| CPU0 7067746ms: 100.0000 MT: 5072736 MF: 3776448
01:12:17.553| ** Global Summary ** R3 CPS=45 [7068312ms] {91 calls in 1887ms => 48.225 CPS}
01:12:17.553| ims_rereg 00 0= 24307 S+F= 24282 F= 3148 E= 12.96434%
01:12:17.553| ims_reg 01 0= 23986 S+F= 23979 F= 470 E= 1.96005%
01:12:17.553| ims_uac 02 0= 81134 S+F= 80199 F= 7821 E= 9.75199%
01:12:17.553| ims_dereg 04 0= 4135 S+F= 4131 F= 453 E= 10.96587%
01:12:17.553| ims_msgc 05 0= 48635 S+F= 48549 F= 3037 E= 6.25554%
01:12:17.553| **ALL** 0= 182197 S+F= 181140 F= 14929 E= 8.24169%
01:12:17.553| SIPP_IHS: ** R3 FINAL IHS (40 CPS)** [7068312ms]
01:12:17.553| ims_rereg 00 (S+F)= 5264 F= 3144 IHS= 59.72644% >= 10.00000%
01:12:17.553| ims_reg 01 (S+F)= 878 F= 470 IHS= 53.53075% >= 10.00000%
01:12:17.553| ims_uac 02 (S+F)= 17353 F= 7797 IHS= 44.93171% >= 10.00000%
01:12:17.553| ims_dereg 04 (S+F)= 849 F= 453 IHS= 53.35689% >= 10.00000%
01:12:17.553| ims_msgc 05 (S+F)= 10220 F= 3029 IHS= 29.63796% >= 10.00000%
01:12:17.553| **ALL** (S+F)= 34564 F= 14893 IHS= 43.08818%
01:12:17.553| Over IHS detected ** STOP NOW **
01:12:34.484| shutdown
01:12:34.484| Closing CPU0 connection...
01:12:34.484| please wait...
01:12:36.486| Manager exit with rc=0
[root@localhost ims_bench]# ./manager -f ims_bench_1/manager.xml

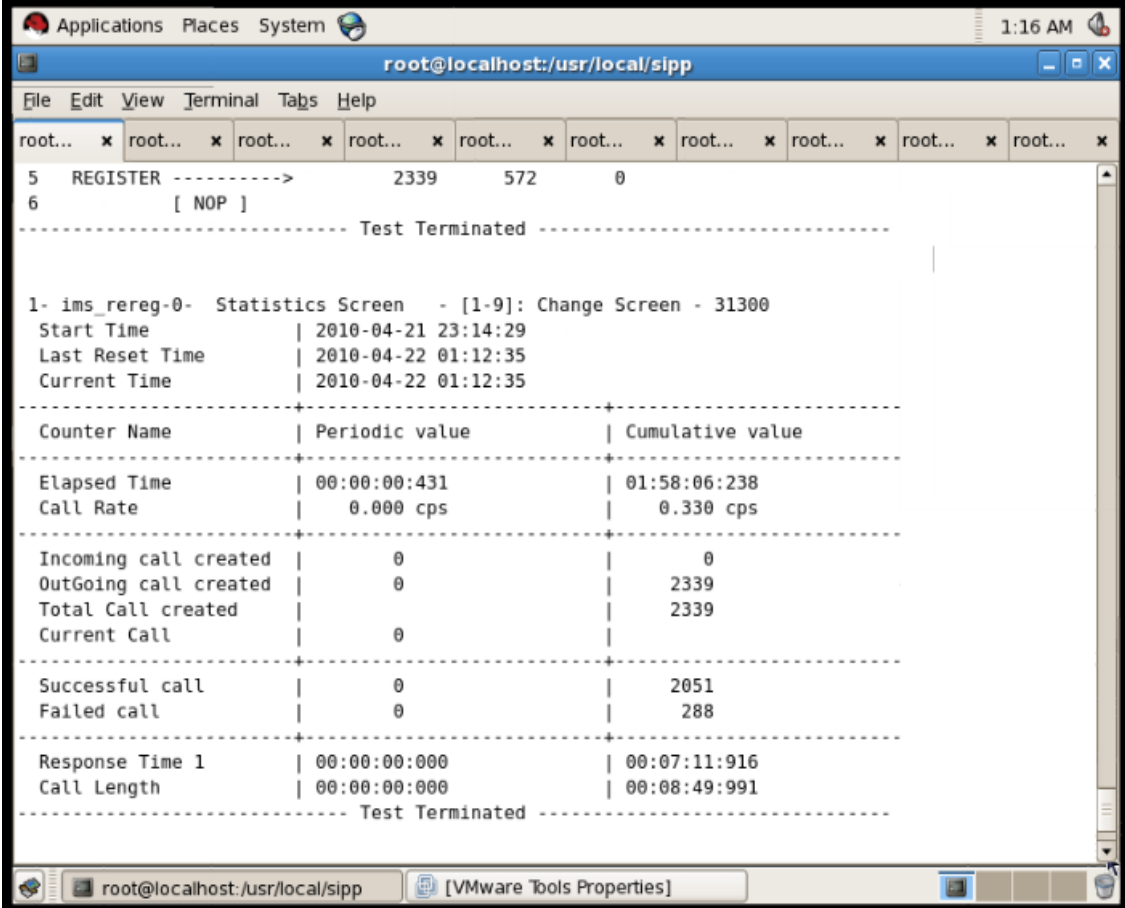
```

**Figure 3-5 Starting Manager on static IP address 10.20.106.106**

Also the naming convention of the SIP subscribers should be cross checked with the database on SUT and configure the naming of subscribers similarly during the configuration of the IMS-Bench (when the IMS-Bench configuration was generated). The SIP subscriber names should look like this “subs003544” in the database on SUT. In order to have this kind of naming for SIP subscribers, the name of the SIP subscribers should be appended with the string “subs” but not “subscriber”, when IMS-Bench configuration was generated. The detailed instructions for installing SW, configuring and starting the Manager are described in detail in APPENDIX C. The detailed instructions for creating IMS-Bench configuration are described in APPENDIX C. The detailed instructions to prepare the Test Systems (TS) for benchmark execution are described in detail in APPENDIX C.

### 3.1.3 Configuring and Starting UAC

The UAC is already configured with the static IP address “10.20.106.107”. The execution details of the UAC are shown in Figure 3-6.



```

root@localhost:/usr/local/sipp
File Edit View Terminal Tabs Help
root... x root... x root... x root... x root... x root... x root... x root... x root... x root... x
5 REGISTER ----->          2339    572    0
6      [ NOP ]
----- Test Terminated -----

1- ims_rereg-0- Statistics Screen - [1-9]: Change Screen - 31300
Start Time           | 2010-04-21 23:14:29
Last Reset Time      | 2010-04-22 01:12:35
Current Time         | 2010-04-22 01:12:35
-----
Counter Name         | Periodic value       | Cumulative value
-----
Elapsed Time         | 00:00:00:431         | 01:58:06:238
Call Rate            | 0.000 cps            | 0.330 cps
-----
Incoming call created | 0                    | 0
OutGoing call created | 0                    | 2339
Total Call created   |                      | 2339
Current Call         | 0                    |
-----
Successful call      | 0                    | 2051
Failed call          | 0                    | 288
-----
Response Time 1     | 00:00:00:000         | 00:07:11:916
Call Length         | 00:00:00:000         | 00:08:49:991
----- Test Terminated -----

```

Figure 3-6 Starting the first UAC on static IP address 10.20.106.107

#### Logging into the UACs with Virtual IP Addresses

It is possible to login to the Virtual IP (for example 10.20.106.2) address of each of the UACs running on the Virtual IP addresses by first login to the UAC with physical IP address (10.20.106.107) and then from that physical host. Each of the UACs should be started from the directory “/usr/local/sipp”. Start all the UAC (clients) in different terminals with the corresponding script namely “run\_x.sh”. To start the first UAC, start the script by name “run\_1.sh”, to start the second UAC, login to the concerned host and then start the script by name “run\_2.sh”. The various phases in establishing a SIP call happens via the SUT, which creates the load on the SUT. The registration and calling phases of SIP subscribers gets initiated between different SIP subscribers.



### 3.1.4 Report Generation at Manager

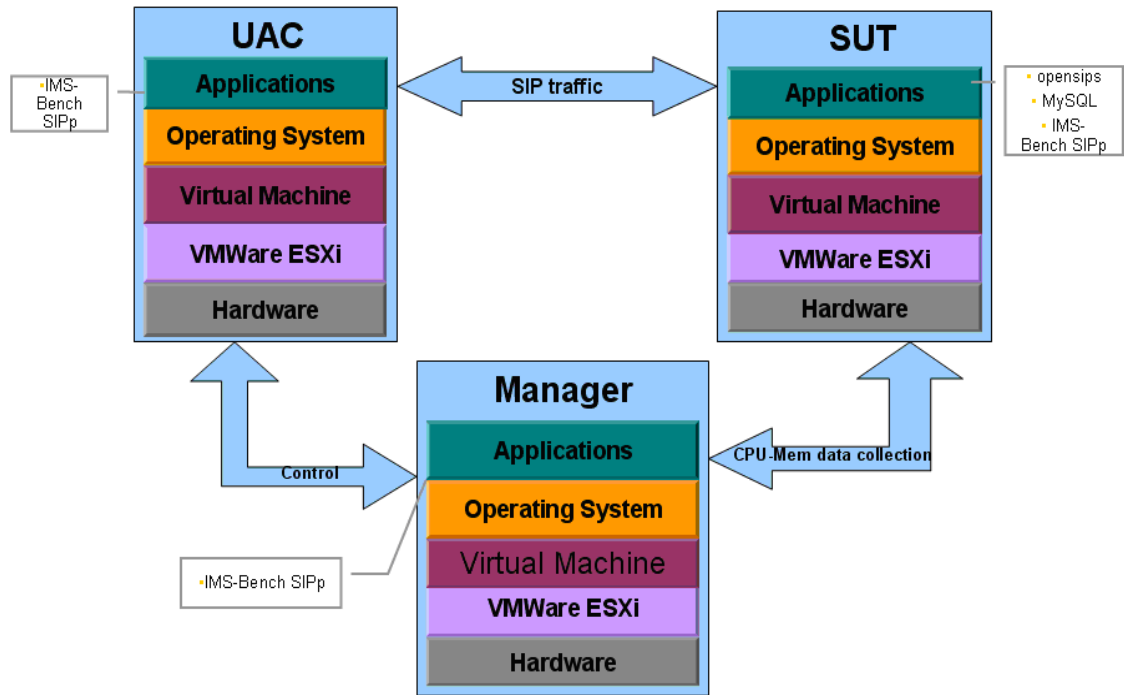
The report generation for the tests executed is carried out on the Manager. The following steps are involved in report generation.

1. Copy the directory "ims\_bench" to some other directory.
2. Execute the command `./scripts/getResults.pl` to fetch the results collected during the benchmark run
3. There will be some unnecessary data in the result files, which are usually large numbers. These large numbers should be removed from the end of the files "sipp.csv" and "sipp\_retrans.csv".
4. The reports are generated with the command `./scripts/doReport.pl -i ims_bench_1/ims_bench.xml`, based on the result files.
5. The actual report of the bench mark run is in the HTML (Hyper Text Markup Language) format, which can be viewed by opening the file "report.html".

It is worth noticing that the data collected is in the format of "csv" files. But the final report was in the HTML format, which helps to present the data in a user friendly form. The IMS-Bench SW has the necessary scripts to generate the report in HTML format. A sample test report generated (after the completion of the experiment) in "Virtual Environment" is provided in the APPENDIX E.

## 3.2 Virtual Environment

Figure 3-7 shows the logical architecture of how the components of the benchmark system fit together. Each of the physical hosts in the virtual environment contains the HW (Intel Xeon X5460), virtualization layer (VMWare ESXi), virtual machine installed with OS (RHEL-5.5) and applications deployed on top of the OS (RHEL-5.5). Three physical hosts (UAC, SUT, Manager) are needed to carry out the experiments in virtual environment.



**Figure 3-7 Experiment setup in Virtual Environment**

The “VMWare vCenter Server” and “vSphere Client” are installed on a physical host, which is installed with the “Microsoft Windows Server 2003” OS. The “vCenter Server” allows to centrally manage the hosts (UAC, SUT, Manager), and enables the use of advanced features such as VMware Distributed Resource Scheduler (DRS), VMware High Availability (HA), and VMware vMotion. The “vSphere Client” allows to connect to the “vCenter Server”, which enables to manage the cluster of hosts together. Each of the ESXi hosts is connected to the SAN (Storage Area Network) to enable the VMs to utilize the bigger storage area. Each of the ESXi hosts is configured with a static IP address. The VMs created on these ESXi hosts are also configured with static IP addresses.

### Cluster Setup

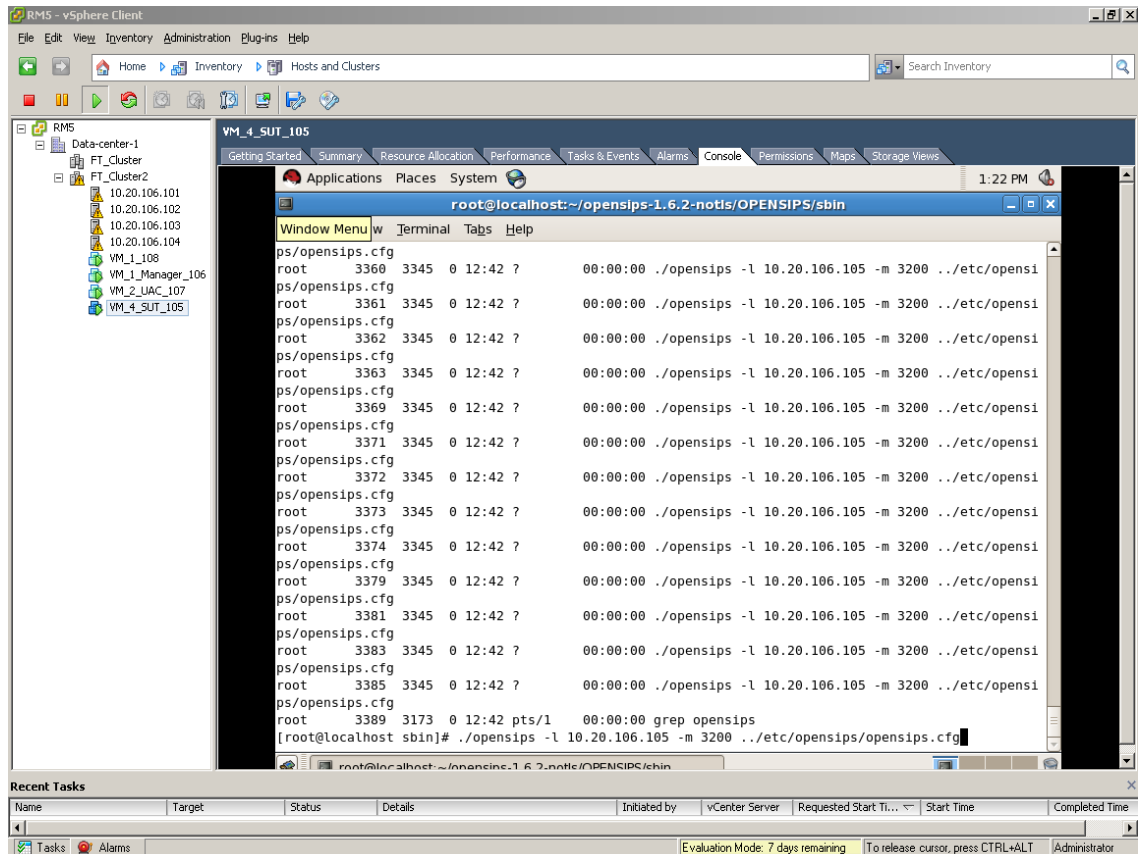
The cluster environment is as shown in Figure 3-11. The cluster environment is needed to carry out the tests in the “Virtual Environment”. The cluster environment can be created and configured using the “vSphere Client” as follows. The steps described below should be followed in the proper order to create and configure the cluster environment.

1. The “Data Center” namely “Data-center-1” needs to be created as a first step.
2. The “Cluster” namely “FT\_Cluster” is created under the “Data-center-1”.
3. The “VMWare ESXi” platform was installed on four physical hosts and are configured with the dedicated IP addresses 10.20.106.101, 10.20.106.102, 10.20.106.103, 10.20.106.104 respectively. Three VMs representing the Manager, SUT and UAC are created on each of these hosts. The “Secondary VM” of SUT gets created on the fourth physical host, when the fault-tolerance feature was turned on. The four hosts are added to the “FT\_Cluster”.
4. The name of the VM representing the SUT is “VM\_4\_SUT\_105”. This VM is installed with the OS namely RHEL(RedHat Enterprise Linux). This VM is assigned with the IP address 10.20.106.105 and is running on the host 10.20.106.104.
5. The name of the VM representing the UAC is “VM\_2\_UAC\_107”. This VM is installed with the OS namely RHEL(RedHat Enterprise Linux). This VM is assigned with the IP address 10.20.106.107 and is running on the host 10.20.106.102
6. The name of the VM representing the Manager is “VM\_1\_Manager\_106”. This VM is installed with the OS namely RHEL(RedHat Enterprise Linux). This VM is assigned with the IP address 10.20.106.106 and is running on the host 10.20.106.101
7. The features such as HA, vMotion are turned on at the cluster level

The procedure to carry out the experiment is the same in all the three types of environments namely “Native Environment”, “Virtual Environment” and “Virtual Environment with FT switched ON”. The detailed instructions about how to carry out the experiment are described in detail in Chapter 3. The detailed instructions about how to install the prerequisite SW and the configuration changes that need to be performed are also described in detail in Chapter 3. The detailed instructions about the execution of the experiment are described in the Chapter 3.1.

### 3.2.1 Configuring and Starting SUT

The SUT is already configured with the static IP address “10.20.106.105”. The command to start the SIP server on the SUT is shown in Figure 3-8. The detailed instructions are described in Chapter 3.1.1.

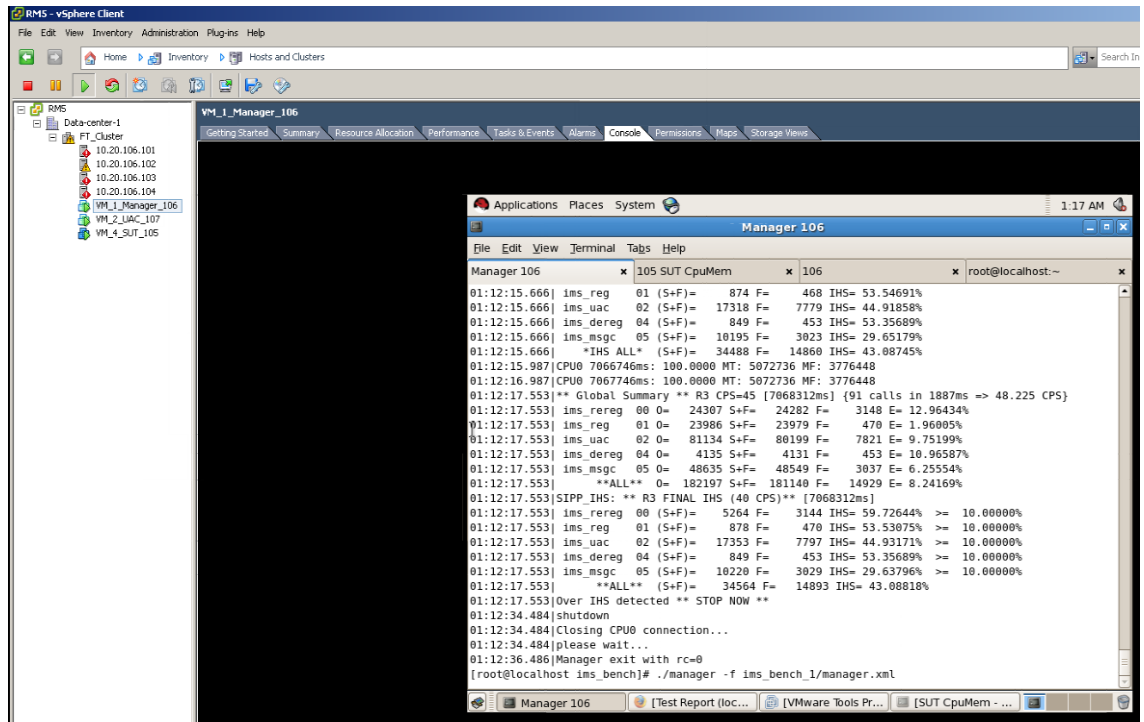


**Figure 3-8 Starting opensips on SUT on VM\_4\_SUT\_105**

The SUT was installed with the MySQL database. The “opensips” SW contains the required scripts to populate the MySQL database with the information of SIP subscribers. The detailed instructions to install the MySQL database and the installation of other software components such as “opensips” is described in detail in the APPENDIX B. The detailed instructions to populate the MySQL database with SIP subscriber information is described in detail in the APPENDIX B. The detailed instructions to start the SIP server (opensips) is described in detail in the APPENDIX B.

### 3.2.2 Configuring and Starting Manager

The Manager is already configured with the static IP address “10.20.106.106”. The command to start the Manager is as shown in Figure 3-9.

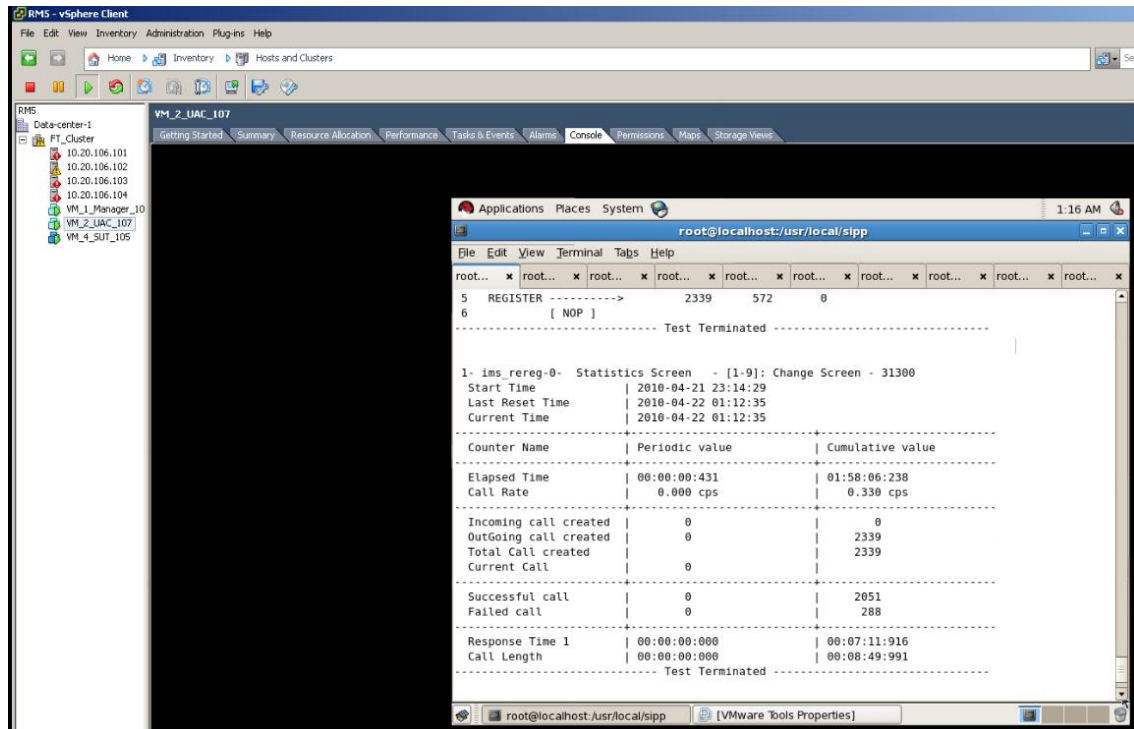


**Figure 3-9 Starting Manager on VM\_1\_Manager\_106**

Figure 3-9 shows the cluster view in the vSphere client. The terminal window on the right hand side of Figure 3-9 shows VM representing the Manager. The detailed instructions are described in Chapter 3.1.2. The instructions to install the SW and to start the Manager are the same in all the three types of environments namely “Native Environment”, “Virtual Environment” and “Virtual Environment with FT switched ON”.

### 3.2.3 Configuring and Starting UAC

The UAC is already configured with the static IP address “10.20.106.107”. The execution details of the UAC are shown in Figure 3-10.



**Figure 3-10 Starting UAC on VM\_2\_UAC\_107**

Figure 3-10 shows the cluster view in the vSphere client. The terminal window on the right hand side of Figure 3-10 shows the VM representing the UAC. The detailed instructions are described in the Chapter 3.1.3. The instructions to install the SW and to start the UAC are the same in all the three types of environments namely “Native Environment”, “Virtual Environment” and “Virtual Environment with FT switched ON”.

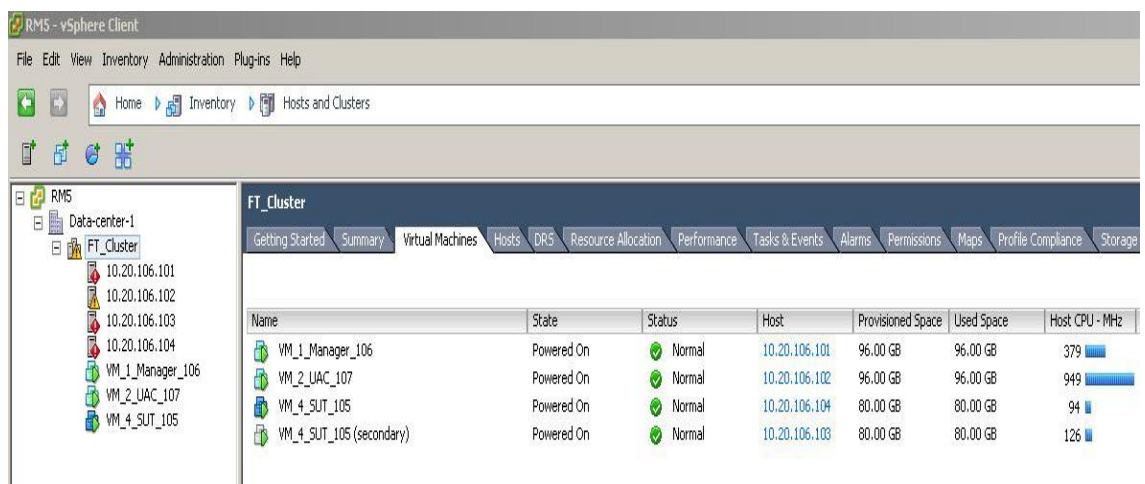
### 3.2.4 Report Generation at Manager

The report generation for the tests executed is carried out on the Manager. The detailed instructions are described in Chapter 3.1.4. It is worth noticing that the data collected was in the format of “csv” files. But the final report was in the HTML format, which helps to present the data in a user friendly manner. The IMS-Bench SW has the necessary scripts to generate the report in HTML format. A sample test report generated (after the completion of the experiment) in “Virtual Environment” is provided in the APPENDIX E.

### 3.3 Virtual Environment with FT Switched ON

The procedure to carry out the experiment is the same in all the three types of environments namely “Native Environment”, “Virtual Environment with FT switched OFF” and “Virtual Environment with FT switched ON”.

The detailed instructions as to how to carry out the experiment are described in detail in Chapter 3.2. The detailed instructions as to how to install the prerequisite SW and the configuration changes that need to be performed are described in Chapter 3.1. The detailed instructions about the execution of the experiment are described in Chapter 3.2.



**Figure 3-11 VMWare-vSphere Cluster view with FT switched on**

The “FT” feature can be switched on through “vSphere Client”. Right click on the “FT\_Cluster” and select the FT from the menu. When the FT is switched on at the SUT, the secondary SUT is created on another physical host. When the failover situation occurs, the secondary SUT takes over the responsibilities from the primary SUT seamlessly and also creates another secondary SUT on another host.

- The VM namely “VM\_4\_SUT\_105” is the SUT assigned with the IP address 10.20.106.105 and running on the host 10.20.106.104
- The VM namely “VM\_4\_SUT\_105(secondary)” is the backup of SUT running on host 10.20.106.103
- The VM namely “VM\_2\_UAC\_107” is the UAC assigned with the IP address 10.20.106.107 and running on host 10.20.106.102

- The VM namely "VM\_1\_Manager\_106" is the Manager assigned with the IP address 10.20.106.106 and running on the host 10.20.106.101

Figure 3-11 shows the cluster view in the vSphere client on the left hand side. Figure 3-11 shows all the VMs on the right hand side. The VMs represent SUT (Primary), SUT (Secondary), UAC and Manager respectively.

The Chapter presented the SW,HW requirements, configuring and starting SUT, Manager and UAC, procedure to carry out the experiment in the three different environments namely " Native Environment", "Virtual Environment" and "Virtual Environment with FT switched ON". The instructions to execute the experiment are the same in all the three types of environments namely "Native Environment", "Virtual Environment" and "Virtual Environment with FT switched ON".



## 4 RESULTS AND ANALYSIS

The results obtained from the three different modes are compared. The performance was measured in the three different modes with respect to the SAPS (Scenario Attempts Per Seconds) as shown in Table 4-1. The scenarios are various phases during the SIP call setup, which are defined in the IMS-Bench configuration.

	Native Environment	Virtual Environment	Virtual Environment with FT turned ON
Design Objective Capacity (SAPS)	139	100	34

**Table 4-1 Performance in three different environments**

The number of SAPS in the virtual environment is lower than that of native environment. The results should not be compared merely on the basis of performance achieved in terms of SAPS as there are huge benefits in using the virtual environments such as effective utilization of resources and other features such as HA, FT, which are readily available without any additional costs.

### 4.1 Performance in Native Environment

The following graphs represent the result of a benchmark run performed by "IMS Bench SIPp", an implementation of the IMS/NGN Performance Benchmark suite. The test reports generated in all of the three environments look quite similar except that the number of SAPS are different. A sample test report generated (after the completion of the experiment) in "Virtual Environment" is provided in the APPENDIX E. Only the measurements pertaining to SUT are discussed in this study even though the test report shows the measurements pertaining to UAC and Manager.

Table 4-2 shows the average of the key measurements for each step of the test. Each of the steps is characterized by the requested load, the effective load, the global IHS (total of all IHS for this step divided by number of SAPS for this step), the scenario IHS

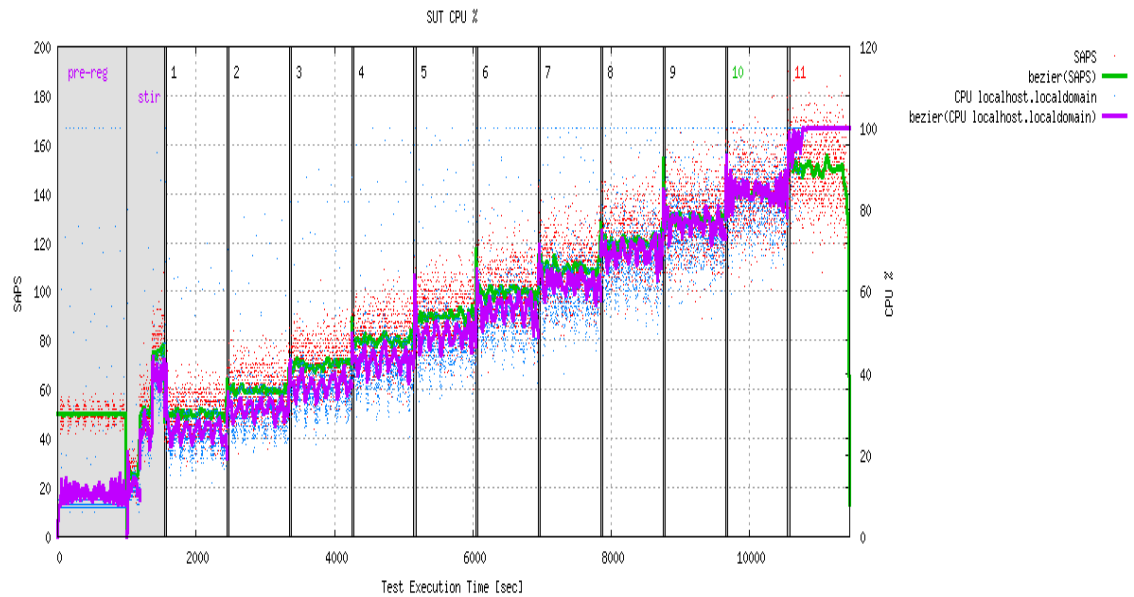
(number of IHS for this step divided by the number of SAPS for this step), the CPU utilization and the available Memory on the SUT.

	Pre-registration	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Step 10	Step 11
Requested load	50	50	60	70	80	90	100	110	120	130	140	150
Effective Load	49.76	50.11	59.75	70.22	80.10	90.07	99.98	109.67	120.28	129.78	139.75	148.63
Ratio ims_rereg %	0.00	15.02	15.05	15.00	14.67	14.83	14.97	14.96	14.92	14.97	15.24	15.27
Ratio ims_reg %	100.00	2.49	2.47	2.47	2.44	2.55	2.39	2.50	2.44	2.55	2.54	2.52
Ratio ims_uac %	0.00	49.84	50.15	49.80	50.17	50.17	50.22	50.02	49.99	50.01	49.63	49.13
Ratio ims_dereg %	0.00	2.53	2.51	2.46	2.36	2.60	2.44	2.47	2.59	2.51	2.53	2.54
Ratio ims_msgc %	0.00	30.12	29.82	30.27	30.36	29.85	29.98	30.05	30.05	29.96	30.05	30.55
CPU localhost.localdomain	10.44	26.19	31.24	37.09	43.07	49.50	55.73	62.17	69.61	76.72	84.27	99.03
Memory localhost.localdomain	2192.86	2172.21	2151.01	2125.67	2096.19	2062.55	2025.37	1984.15	1938.95	1889.03	1835.95	1779.15
IHS ims_rereg %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	20.92
IHS ims_reg %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	18.53
IHS ims_uac %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.14	48.08
IHS ims_dereg %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	18.24
IHS ims_msgc %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	31.48
global IHS %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.08	37.36

**Table 4-2 Resource utilization of SUT in Native Environment**

The available Memory is expressed in Mega Bytes, and the requested and effective loads in Scenarios Attempts Per Seconds (SAPS). It should be noted that the IHS percentages represented in this table are the number of failures for a step divided by the number of scenario attempts for this step, and so is not the average of (IHS per seconds). The number of SAPS were maintained as 50 (constant load) during the “pre-registration” phase. The number of SAPS (load) were increased gradually from step-1 to step-11 by incrementing the number of SAPS by 10 in each step. The utilization of the CPU resources of the SUT were also increasing gradually with the increase in the number of SAPS in each step. The resources of the CPU had got exhausted completely in the step-11.

Graph 4-1 represents the percentage utilization of the CPU of the SUT over the time and the SAPs for each of the steps. One of the curves shown in the graph represents the SAPs and the other curve represents the percentage utilization of the CPU of the SUT over the time for each of the steps. In Figure 4-1, the SAPs are represented on y-axis on the left hand side, where as the CPU utilization is represented on y-axis on the right hand side.



**Graph 4-1 CPU utilization of SUT in Native Environment**

It can be observed from the graph that the experiment was initially carried out with a sample load in the “pre-reg” and “stir” phases, before starting with the testing of the actual load (SAPS). The number of SAPS were increased gradually from iteration-1 to iteration-11. The resources of the CPU had got exhausted completely in the iteration-11. The number of SAPS at this stage were 139, which was the maximum measurement of SAPS achieved in the “native environment”. The entire process gets carried out automatically once the test was started at the Manager.

## 4.2 Performance in Virtual Environment

The following graphs represent the result of a benchmark run performed by "IMS Bench SIPp", an implementation of the IMS/NGN Performance Benchmark suite. The test reports generated in all of the three environments look quite similar except that the number of SAPS are different. A sample test report generated (after the completion of the experiment) in “Virtual Environment” is provided in the APPENDIX E. Only the measurements pertaining to SUT are discussed in this study even though the test report shows the measurements pertaining to UAC and Manager.

Table 4-3 shows the average of the key measurements for each step of the test. Each of the steps is characterized by the requested load, the effective load, the global IHS (total of all IHS for this step divided by number of SAPS for this step), the scenario IHS (number of IHS for this step divided by the number of SAPS for this step), the CPU utilization and the available Memory on the SUT.

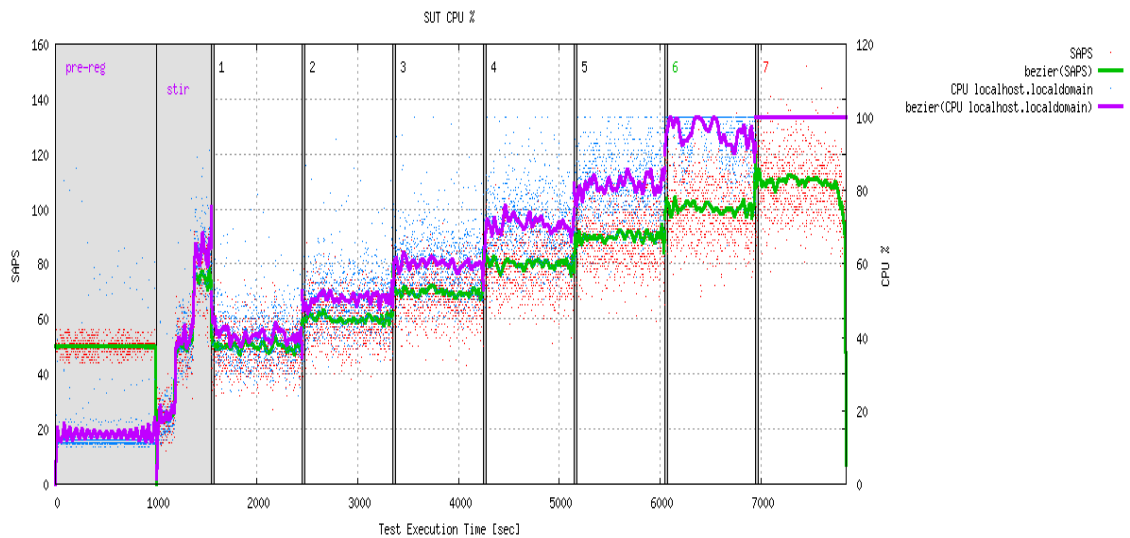
	Pre-registration	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
Requested load	50	50	60	70	80	90	100	110
Effective Load	50.00	50.03	60.09	69.67	80.12	89.87	100.44	109.00
Ratio ims_rereg %	0.00	14.89	15.12	15.05	15.20	14.98	15.06	14.94
Ratio ims_reg %	100.00	2.52	2.46	2.38	2.42	2.47	2.49	2.42
Ratio ims_uac %	0.00	49.81	50.12	49.75	49.94	50.21	49.85	49.41
Ratio ims_dereg %	0.00	2.46	2.36	2.54	2.57	2.38	2.55	2.58
Ratio ims_msgc %	0.00	30.32	29.93	30.27	29.87	29.96	30.06	30.66
CPU localhost.localdomain	13.57	40.63	50.49	60.10	70.75	81.59	95.90	99.99
Memory localhost.localdomain	3776.43	3755.47	3735.19	3710.63	3682.42	3649.50	3613.57	3571.89
SIPP CPU localhost.localdomain	0.55	0.66	0.73	0.82	0.99	1.18	1.33	1.72
SIPP MEM localhost.localdomain	13977.39	13958.20	13957.31	13953.85	13949.01	13943.69	13937.41	13927.22
IHS ims_rereg %	0.00	0.00	0.00	0.00	0.00	0.00	3.49	21.25
IHS ims_reg %	0.00	0.00	0.00	0.00	0.00	0.00	1.75	13.57
IHS ims_uac %	0.00	0.00	0.00	0.00	0.00	0.19	7.10	43.16
IHS ims_dereg %	0.00	0.00	0.00	0.00	0.00	0.00	1.75	13.93
IHS ims_msgc %	0.00	0.00	0.00	0.00	0.00	0.00	4.83	29.67
global IHS %	0.00	0.00	0.00	0.00	0.00	0.09	5.60	34.28

**Table 4-3 Resource utilization of SUT in Virtual Environment**

The available Memory is expressed in Mega Bytes, and the requested and effective loads in Scenarios Attempts Per Seconds (SAPS). It should be noted that the IHS percentages represented in this table are the number of failures for a step divided by the number of scenario attempts for this step, and so is not the average of (IHS per seconds). The number of SAPS were maintained as 50 (constant load) during the “pre-registration” phase. The number of SAPS (load) were increased gradually from step-1 to step-7 by incrementing the number of SAPS by 10 in each step. The utilization of the CPU resources of the SUT were also increasing gradually with the increase in the number of SAPS in each step. The resources of the CPU had got exhausted completely in the step-7.

Graph 4-2 represents the percentage utilization of the CPU of the SUT over the time and the SAPs for each of the steps. One of the curves shown in the graph represents the SAPs and the other curve represents the percentage utilization of the CPU of the SUT over the time for each of the steps. In Figure 4-2, the SAPs are represented on y-

axis on the left hand side, where as the CPU utilization is represented on y-axis on the right hand side.



**Graph 4-2 CPU utilization of SUT in Virtual Environment**

It can be observed from the graph that the experiment was initially carried out with a sample load in the “pre-reg” and “stir” phases, before starting with the testing of the actual load (SAPS). The number of SAPS were increased gradually from step-1 to step-7. The resources of the CPU had got exhausted completely in the step-7. The number of SAPS at this stage were 100, which was the maximum measurement of SAPS achieved in the “virtual environment”. The entire process gets carried out automatically once the test was started at the Manager.

### 4.3 Performance in Virtual Environment with FT Switched ON

The following graphs represent the result of a benchmark run performed by "IMS Bench SIPp", an implementation of the IMS/NGN Performance Benchmark suite. The test reports generated in all of the three environments look quite similar except that the number of SAPS are different. A sample test report generated (after the completion of

the experiment) in “Virtual Environment” is provided in the APPENDIX E. Only the measurements pertaining to SUT are discussed in this study even though the test report shows the measurements pertaining to UAC and Manager.

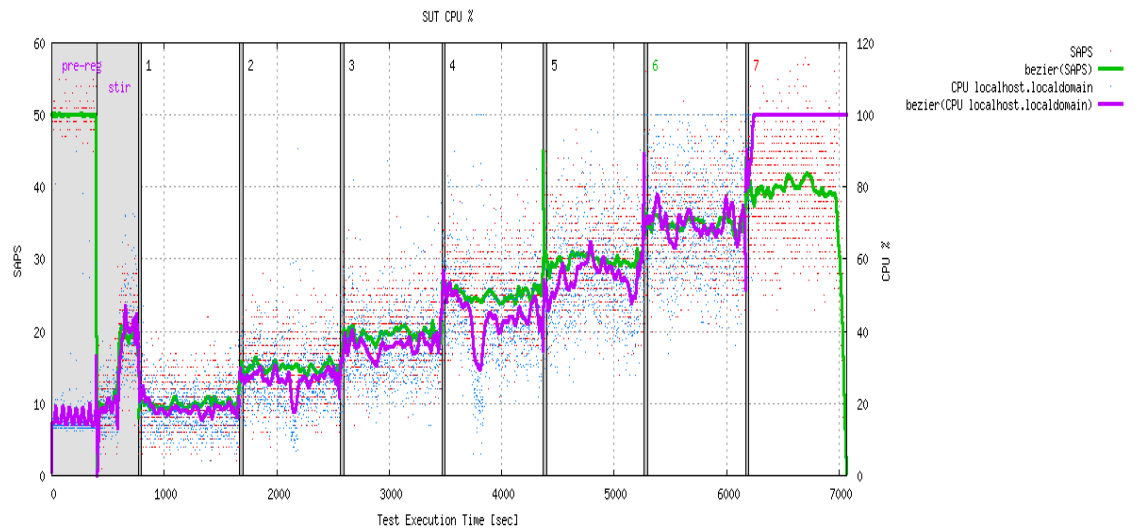
Table 4-4 shows the average of the key measurements for each step of the test. Each of the steps is characterized by the requested load, the effective load, the global IHS (total of all IHS for this step divided by number of SAPS for this step), the scenario IHS (number of IHS for this step divided by the number of SAPS for this step), the CPU utilization and the available Memory on the SUT.

	Pre-registration	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
Requested load	50	10	15	20	25	30	35	40
Effective Load	49.87	9.91	14.92	19.59	25.12	29.83	34.84	38.08
Ratio ims_rereg %	0.00	14.68	14.89	14.67	14.96	15.23	14.98	15.57
Ratio ims_reg %	100.00	2.47	2.55	2.30	2.49	2.37	2.44	2.60
Ratio ims_uac %	0.00	49.45	50.32	50.17	49.72	49.63	50.17	49.03
Ratio ims_dereg %	0.00	2.80	2.56	2.45	2.43	2.58	2.75	2.52
Ratio ims_msgc %	0.00	30.60	29.68	30.42	30.40	30.18	29.66	30.27
CPU localhost.localdomain	16.13	18.61	26.69	35.95	44.13	55.34	69.91	98.95
Memory localhost.localdomain	3857.90	3842.02	3837.34	3830.19	3822.19	3811.81	3799.39	3784.15
SIPP CPU localhost.localdomain	0.60	0.42	0.44	0.48	0.54	0.60	0.68	1.08
SIPP MEM localhost.localdomain	13985.61	13978.78	13978.04	13976.08	13974.67	13972.54	13970.26	13964.36
IHS ims_rereg %	0.00	0.00	0.05	0.00	0.09	0.00	0.00	59.67
IHS ims_reg %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	53.60
IHS ims_uac %	0.00	0.00	0.11	0.00	0.16	0.00	0.06	46.63
IHS ims_dereg %	0.00	0.00	0.00	0.00	0.00	0.00	0.00	53.29
IHS ims_msgc %	0.00	0.00	0.00	0.00	0.12	0.00	0.00	29.66
global IHS %	0.00	0.00	0.06	0.00	0.13	0.00	0.03	43.87

**Table 4-4 Resource utilization of SUT in Virtual Environment with FT switched on**

The available Memory is expressed in Mega Bytes, and the requested and effective loads in Scenarios Attempts Per Seconds (SAPS). It should be noted that the IHS percentages represented in this table are the number of failures for a step divided by the number of scenario attempts for this step, and so is not the average of (IHS per seconds). The number of SAPS were maintained as 50 (constant load) during the “pre-registration” phase. The number of SAPS (load) were increased gradually from step-1 to step-7 by incrementing the number of SAPS by 5 in each step. The utilization of the CPU resources of the SUT were also increasing gradually with the increase in the number of SAPS in each step. The resources of the CPU had got exhausted completely in the step-7.

Graph 4-3 represents the percentage utilization of the CPU of the SUT over the time and the SAPs for each of the steps. One of the curves shown in the graph represents the SAPs and the other curve represents the percentage utilization of the CPU of the SUT over the time for each of the steps. In Figure 4-3, the SAPs are represented on y-axis on the left hand side, where as the CPU utilization is represented on y-axis on the right hand side.



**Graph 4-3 CPU utilization of SUT in Virtual Environment with FT switched on**

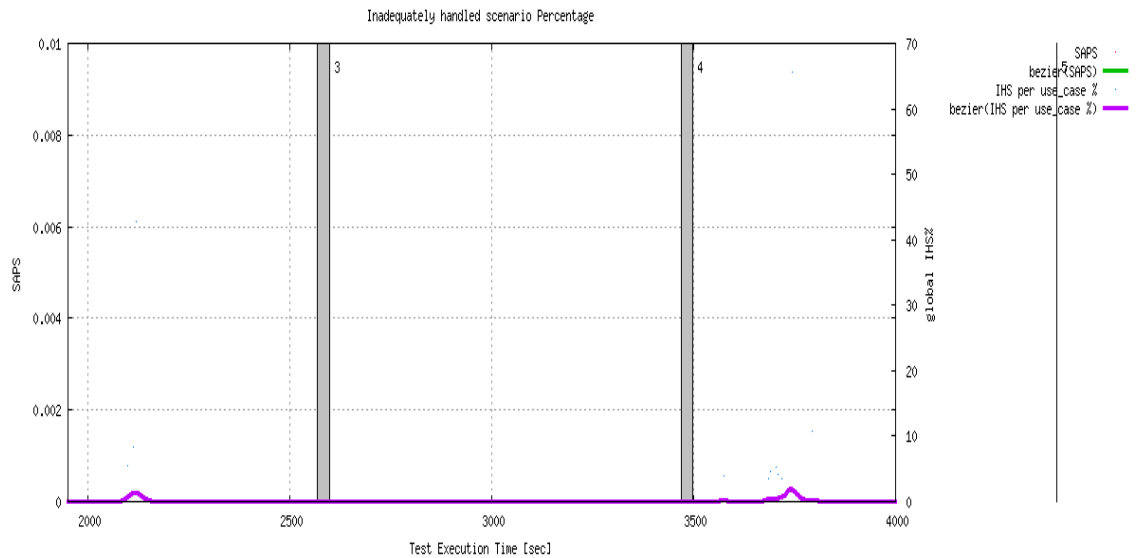
It can be observed from the graph that the experiment was initially carried out with a sample load in the “pre-reg” and “stir” phases, before starting with the testing of the actual load (SAPS). The number of SAPs were increased gradually from step-1 to step-7. The resources of the CPU had got exhausted completely in the step-7. The number of SAPs at this stage were 34, which was the maximum measurement of SAPs achieved in the “virtual environment with FT switched on”.

### Analysis of Failover Situation

The graphs are zoomed in to analyze the failover situation, when the FT is switched ON in the virtual environment. The failover situation was created with the help of the VMWare vSphere client. The failover situations were made to be occurred on the timeline on x-axis between 2000-2500 and 3500-4000 seconds.

## Analysis of Inadequately Handled Scenarios

Graph 4-4 shows the graph representing the global IHS over the time line. The graph was zoomed in to analyze how the IHS versus SAPS varies, when the failover situations were made to be occurred on the timeline on x-axis between 2000-2500 and 3500-4000 seconds. The SAPS are represented on y-axis on a scale of 0 to 0.01 and the global IHS on y-axis on a scale of 0 to 70.

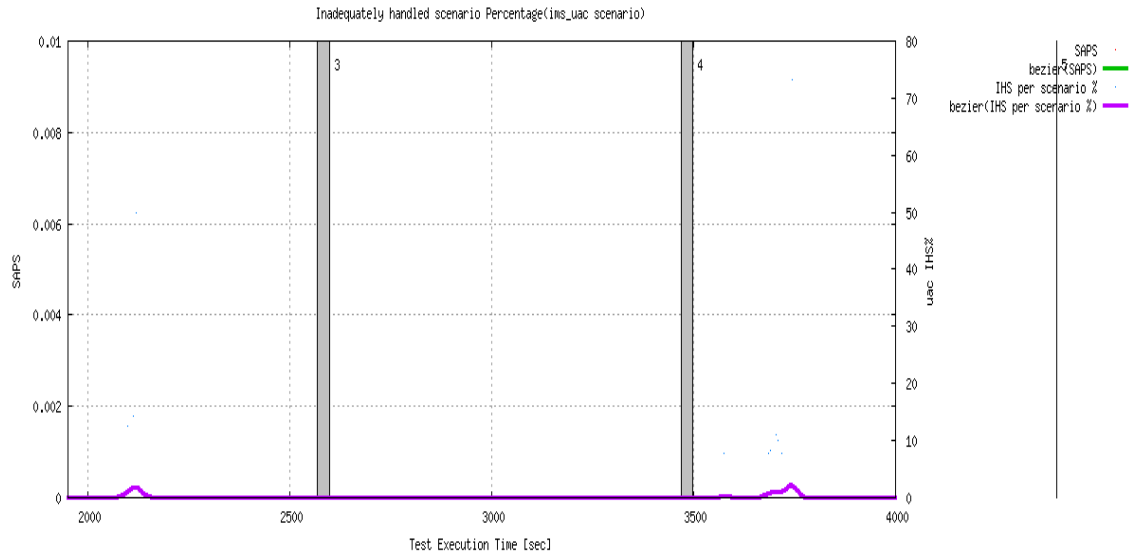


**Graph 4-4 Inadequately Handled Scenarios**

It can be observed from the graph that the effect of the failover is almost negligible as expected. The inadequately handled SAPS are less than 0.02.

Graph 4-5 shows the graph representing the IHS of the ims\_uac scenario over the time line. The graph was zoomed in to analyze how the IHS versus SAPS varies, when the failover situations were made to be occurred on the timeline on x-axis between 2000-2500 and 3500-4000 seconds. The SAPS are represented in on y-axis on a scale of 0 to 0.01 and the IHS on y-axis on a scale of 0 to 80.



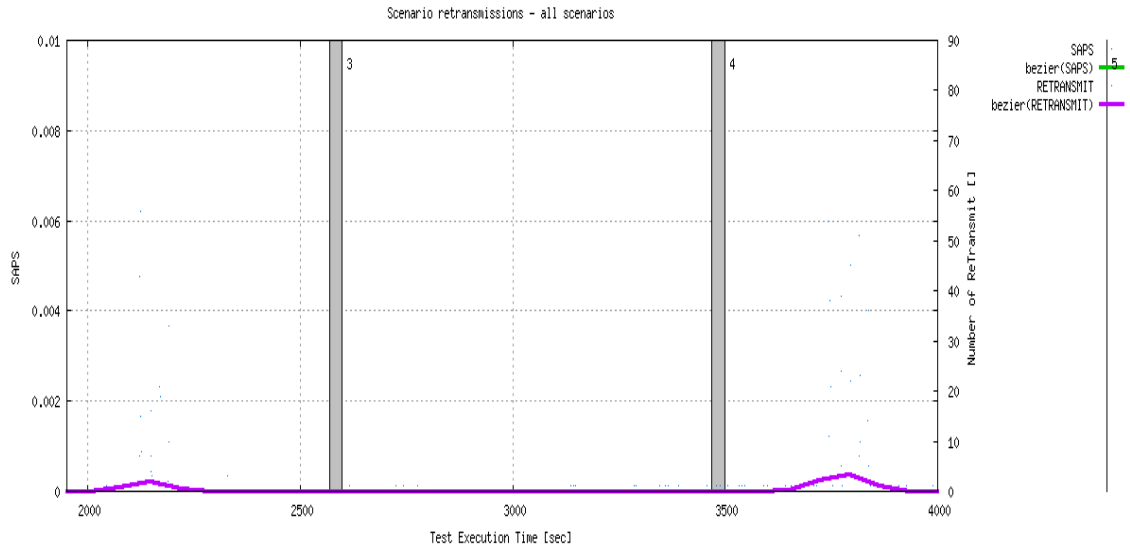


**Graph 4-5 Inadequately Handled Scenarios for ims\_uac scenario**

It can be observed from the graph that the effect of the failover is almost negligible as expected. The inadequately handled SPS are less than 0.02.

### Analysis of Retransmissions

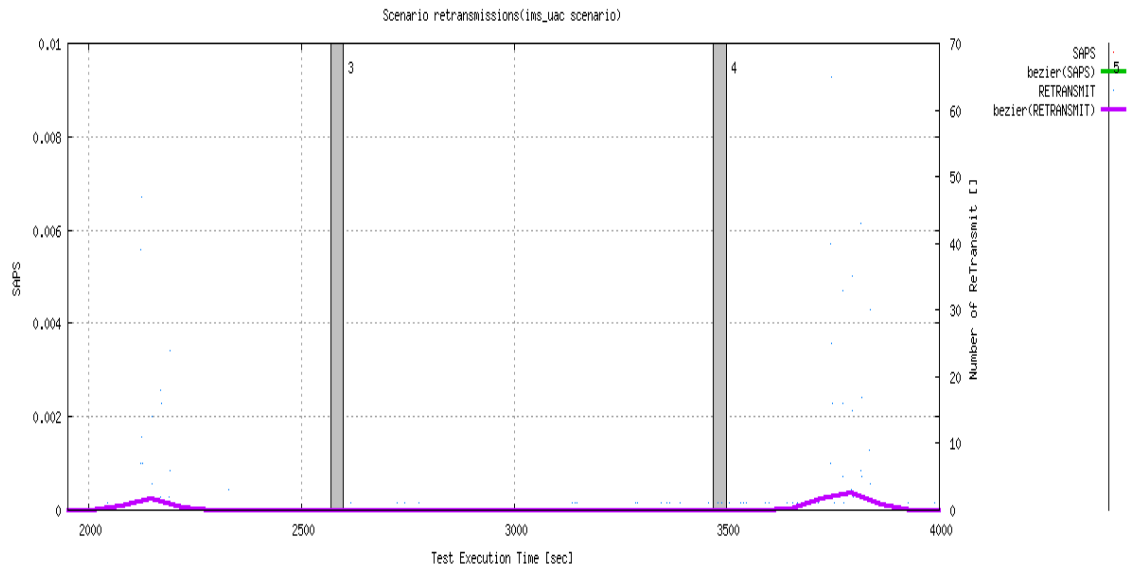
Graph 4-6 shows the graph representing the total number of retransmits versus SPS over the time line. The graph was zoomed in to analyze how the total number of retransmits versus SPS varies, when the failover situations were made to be occurred on the timeline on x-axis between 2000-2500 and 3500-4000 seconds. The SPS is represented in on y-axis on a scale of 0 to 0.01 and the total number of retransmits on y-axis on a scale of 0 to 90.



#### Graph 4-6 Retransmissions for All Scenarios

It can be observed from the graph that the effect of the failover is almost negligible as expected. The scenarios, which were being processed at the time, when the failover situation occurred are usually retransmitted. The total number of retransmitted scenarios is less than 0.02.

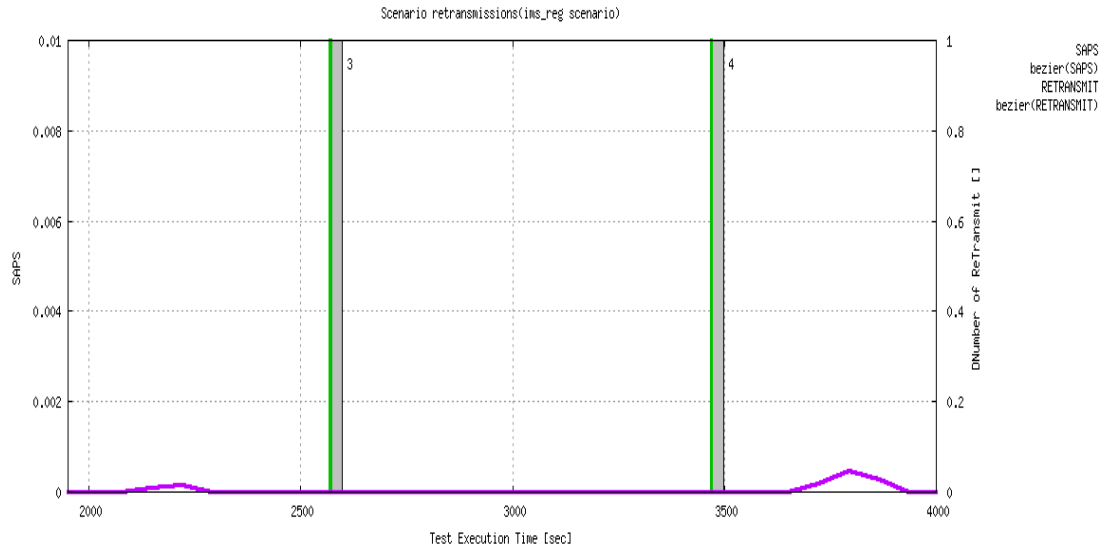
Graph 4-7 shows the graph representing the total number of retransmits for the scenario `ims_uac` versus SAPS over the time line. The graph was zoomed in to analyze how the total number of retransmits for the scenario `ims_uac` versus SAPS varies, when the failover situations were made to be occurred on the timeline on x-axis between 2000-2500 and 3500-4000 seconds. The SAPS is represented in on y-axis on a scale of 0 to 0.01 and the total number of retransmits for the scenario `ims_uac` on y-axis on a scale of 0 to 70.



**Graph 4-7 Retransmissions for ims\_uac scenario**

It can be observed from the graph that the effect of the failover is almost negligible as expected. The scenarios, which were being processed at the time, when the failover situation occurred are usually retransmitted. The total number of retransmitted scenarios for the scenario ims\_uac is less than 0.02.

Graph 4-8 shows the graph representing the total number of retransmits for the scenario ims\_reg versus SAPS over the time line. The graph was zoomed in to analyze how the total number of retransmits for the scenario ims\_reg versus SAPS varies, when the failover situations were made to be occurred on the timeline on x-axis between 2000-2500 and 3500-4000 seconds. The SAPS is represented in on y-axis on a scale of 0 to 0.01 and the total number of retransmits for the scenario ims\_reg on y-axis on a scale of 0 to 1.

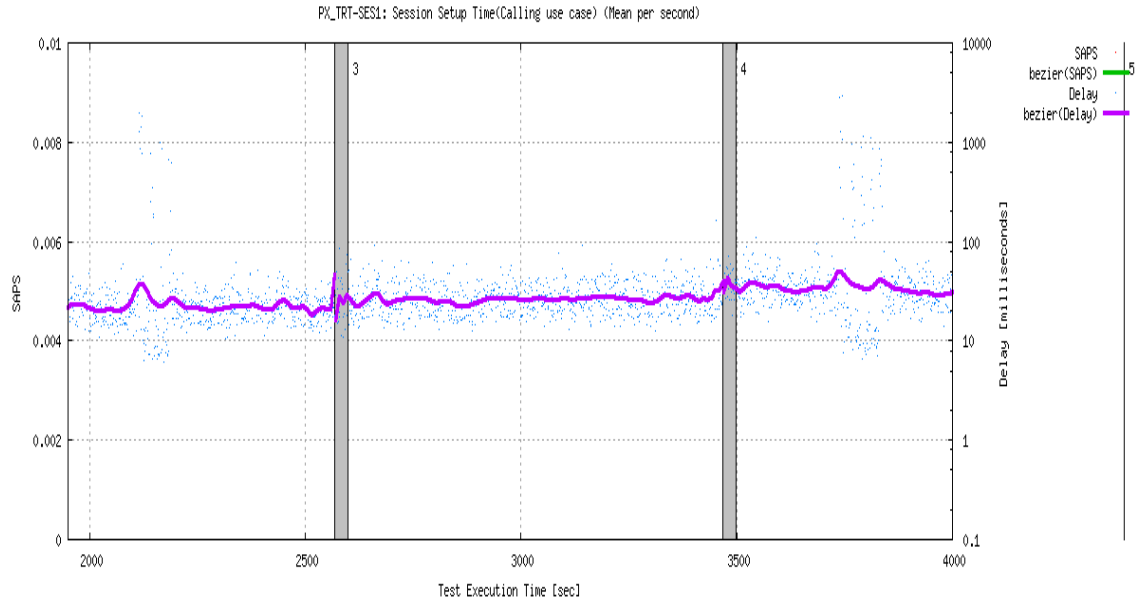


**Graph 4-8 Retransmissions for ims\_reg scenario**

It can be observed from the graph that the effect of the failover is almost negligible as expected. The scenarios, which were being processed at the time, when the failover situation occurred are usually retransmitted. The total number of retransmitted scenarios for the scenario `ims_reg` is less than 0.02.

### Analysis of Calling Use Case

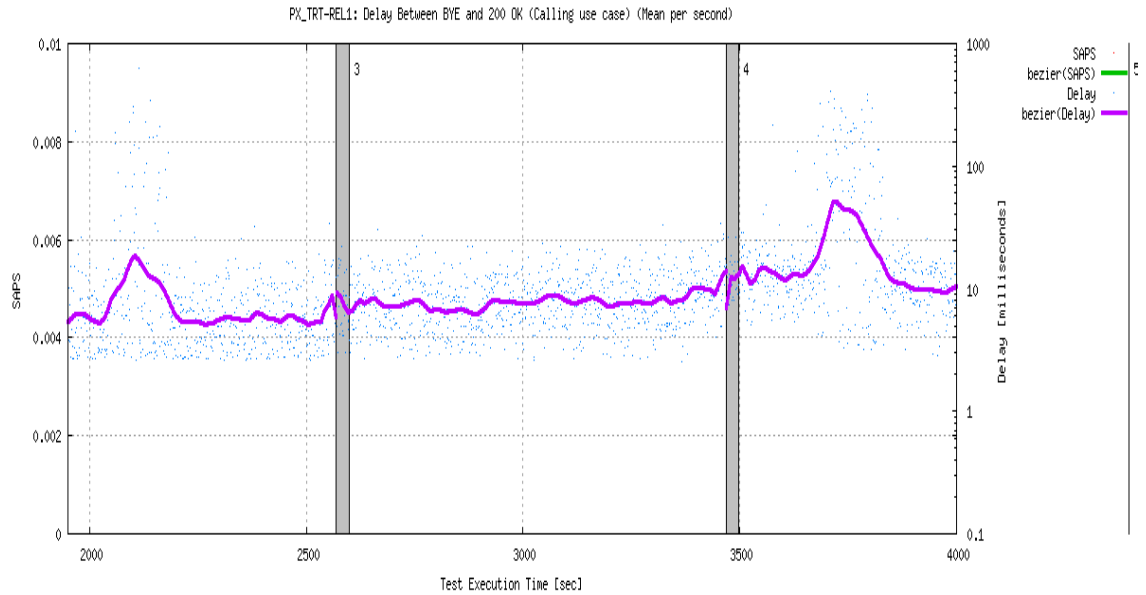
Graph 4-9 shows the graph representing the delay in milliseconds for the calling use case scenario (during the session setup) versus SAPS over the time line. The graph was zoomed in to analyze how the delay that had occurred during the calling use case scenario (during the session setup) versus SAPS varies, when the failover situations were made to be occurred on the timeline on x-axis between 2000-2500 and 3500-4000 seconds. The SAPS are represented on y-axis on a scale of 0 to 0.01 and the delay on y-axis on a scale of 0.1 to 10000 milliseconds.



**Graph 4-9 Calling Use Case for Session Setup**

It can be observed from the graph that the effect of the failover is almost negligible as expected. The calling use case (during the session setup) suffers a negligible delay in the range of 10 to 100 milliseconds and the affected SAPS are in the range of 0.004 to 0.006, which were being processed at the time, when the failover situation occurred.

Graph 4-10 shows the graph representing the delay in milliseconds for the calling use case scenario (during the BYE and OK) versus SAPS over the time line. The graph was zoomed in to analyze how the delay that had occurred during the calling use case scenario (during the BYE and OK) versus SAPS varies, when the failover situations were made to be occurred on the timeline on x-axis between 2000-2500 and 3500-4000 seconds. The SAPS are represented on y-axis on a scale of 0 to 0.01 and the delay on y-axis on a scale of 0.1 to 1000 milliseconds.



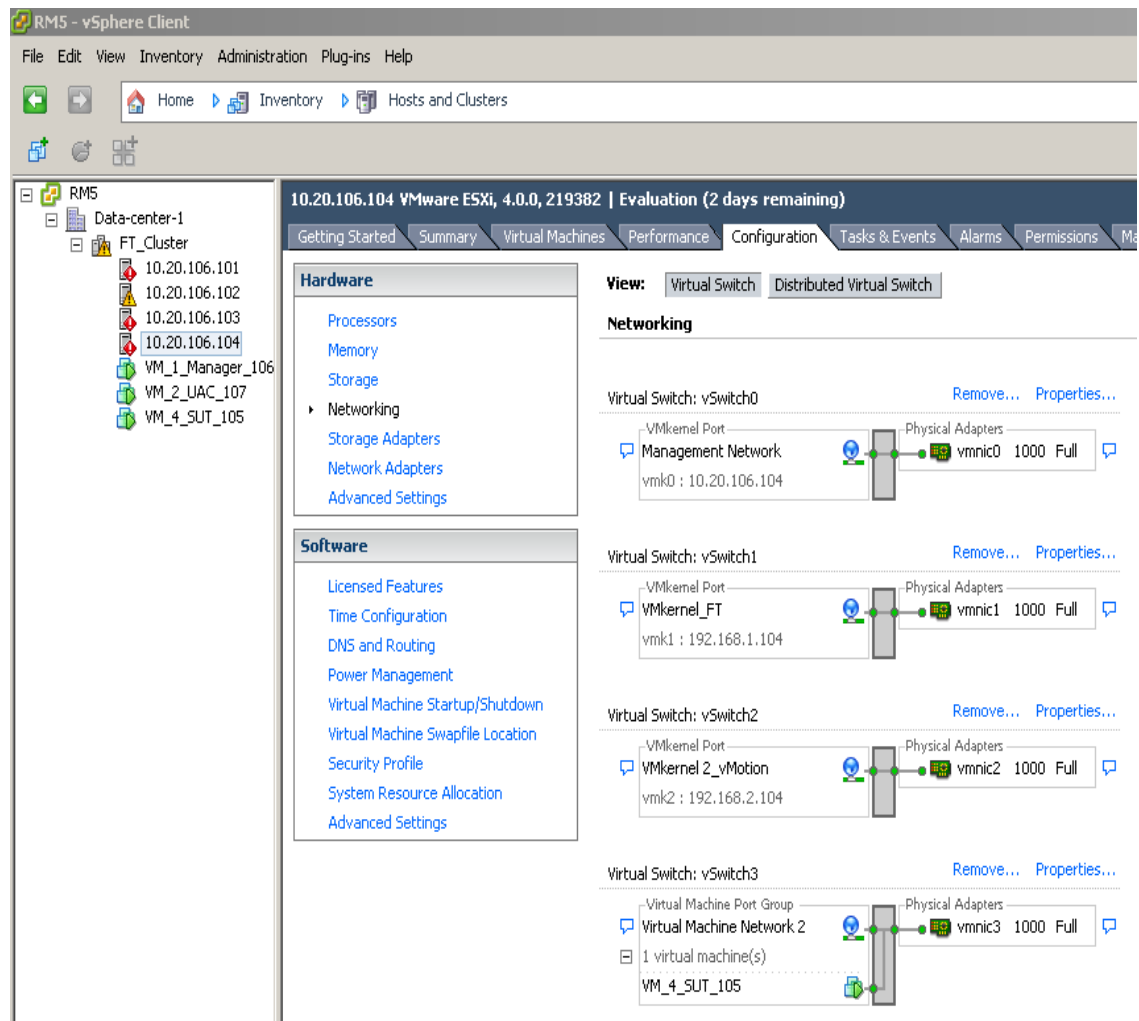
**Graph 4-10 Calling Use Case : Delay between BYE and OK**

It can be observed from the graph that the effect of the failover is almost negligible as expected. The calling use case (during the BYE and OK) suffers a negligible delay in the range of 1 to 100 milliseconds and the affected SPS are in the range of 0.004 to 0.008, which were being processed at the time, when the failover situation occurred.

## 4.4 Performance Improvement Techniques in Virtual Environment

The performance in the virtual environment can be improved to some extent by optimizing the configuration in the virtual environment. Some of the most common optimization techniques are “installation of the VMWare Tools” and “utilization of all the NICs available on the physical host”.

The number of SPS increased from 89 to 100 after configuring and utilizing all the available NICs (Network Interface Cards) as shown in Figure 4-1. Dedicated NICs were configured to carry different types of traffic such as “Management Network traffic”, “FT traffic”, “vMotion traffic” and “Virtual Machine Network traffic” on separate NICs. Otherwise, all the traffic was channeled through a single NIC, which would be a bottleneck to achieve the peak performance.



**Figure 4-1 Configuring all the Network Interface Cards**

It is possible to run the entire setup in one physical host, which is installed with ESXi, by creating four VMs on the same ESXi host. Three VMs can function like SUT, Manager, UAC and the fourth VM can be installed with the “VMWare vCenter Server” and “vSphere Client” for administering the VMs on the ESXi host. This scenario of running all the VMs in a single host would be useful to carry out a sample run of the experiment. It is highly advisable to run at least SUT on a separate physical host to achieve the more practical results.

This Chapter presented the analysis of results with respect to the performance in “Native environment”, “Virtual environment” and “Virtual Environment with FT switched ON”. Then presented the techniques to improve the performance in “Virtual Environment”.

## 5 SUMMARY

This thesis presented an evaluation of the performance of a SIP based application among “Native Environment”, “Virtual Environment” and “Virtual Environment with FT feature turned on”. According to the analysis the effect of failover situation was almost negligible, when the switch over occurs in “Virtual Environment with FT feature turned on”. It was noticed that the configuration of the physical hosts needed to be downgraded to achieve the complete exhaustion of the CPU resources on the System Under Test. The goal of this study was reached as the advantages of Fault-Tolerance feature in “Virtual Environment” was demonstrated and also the performance metrics were also analyzed.

The outcome of the study has been applied at NSN (Nokia Siemens Networks) to carry out the experiments to estimate the influence of virtualization on power consumption. This research was carried out because the power management and optimization of power consumption in telecommunication products is vital. The effects of power consumption in virtual environments is measured since most of the products are moving towards virtualization to achieve the efficient utilization of resources.

This study covered experiments based on the “VMWare ESXi” platform only. There is another competent platform namely Xen, which is an open source product providing the virtualization infrastructure with the “fault tolerance” feature. An attempt was made to carry out the experiment using the Xen platform. But the “fault tolerance” feature in the Xen platform was not fully matured and stable at the time when this experiment was carried out. This experiment can be repeated in the future with an open source based Xen platform and the results can be compared with the results obtained with the “VMWare ESXi” platform.



## 6 REFERENCES

- [1] Hypervisor-Based Fault-Tolerance, <http://www.cs.cornell.edu/fbs/publications/HyperFTol.pdf> (Accessed May 4, 2011)
- [2] Kemari, <http://www.xen.org/community/projects.html> (Accessed May 4, 2011)
- [3] Remus, <http://people.cs.ubc.ca/~andy/papers/remus-nsdi-final.pdf> (Accessed May 4, 2011)
- [4] Fast Memory State Synchronization for Virtualization-based Fault Tolerance, <http://www.ecsl.cs.sunysb.edu/tr/TR235.pdf> (Accessed May 4, 2011)
- [5] VMWare vSphere 4.0 Fault Tolerance, <http://xtravirt.com/xd10007> (Accessed May 4, 2011)
- [6] VMWare vSphere Architecture, [http://www.vmware.com/files/pdf/perf-vsphere-fault\\_tolerance.pdf](http://www.vmware.com/files/pdf/perf-vsphere-fault_tolerance.pdf) (Accessed May 4, 2011)
- [7] An Introduction to Virtualization with VMware vSphere 4, <http://www.petri.co.il/vmware-vsphere-4.htm> (Accessed May 4, 2011)
- [8] Processors and guest OS that support VMware Fault Tolerance, [http://kb.vmware.com/selfservice/microsites/search.do?language=en\\_US&cmd=displayKC&externalId=1008027](http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1008027) (Accessed May 4, 2011)
- [9] Control Plane SW, <http://sourceforge.net/projects/control-plane/> (Accessed May 4, 2011)
- [10] OPENSIPS, <http://opensips.org/pub/opensips/1.6.2/src/> (Accessed May 4, 2011)
- [11] IMS Bench SIPP, [http://sipp.sourceforge.net/ims\\_bench/reference.html](http://sipp.sourceforge.net/ims_bench/reference.html) (Accessed May 4, 2011)
- [12] Xen, <http://wiki.xensource.com/xenwiki/Xen4.0?highlight=%28xen%29%284.0%29> (Accessed May 4, 2011)

[13] UAC User Model,

[http://www.spec.org/specsip/docs/designdocument.html#4\\_4\\_1\\_UAC\\_User\\_Model](http://www.spec.org/specsip/docs/designdocument.html#4_4_1_UAC_User_Model)(Accessed May 4, 2011)

[14] Intel Xeon X5460 Quad Core Processor Specification,

[http://ark.intel.com/Product.aspx?id=33087&code=Intel%20Xeon%20Processor+X5460+\(12M+Cache%2C+3.16+GHz%2C+1333+MHz+FSB\)](http://ark.intel.com/Product.aspx?id=33087&code=Intel%20Xeon%20Processor+X5460+(12M+Cache%2C+3.16+GHz%2C+1333+MHz+FSB)) (Accessed May 4, 2011)

[15] MySQL, <http://www.mysql.com/>

[16] VMWare vSphere Fault-Tolerance,

[http://www.vmware.com/files/pdf/fault\\_tolerance\\_recommendations\\_considerations\\_on\\_vmw\\_vsphere4.pdf](http://www.vmware.com/files/pdf/fault_tolerance_recommendations_considerations_on_vmw_vsphere4.pdf)

**APPENDIX A**

**Specification of the Intel Xeon X5460**

<b>Specification of Intel Xeon X5460</b>	
Processor Number	Intel® Xeon® X5460 Quad Core Processor 3.16 GHz
Processor speed	3.16 GHz
Number of processors	2 processors
Processor core available	Quad
Internal Cache	12 MB (2 x 6 MB) Level 2 cache memory
Standard memory	4 GB (4 x 1 GB) Standard Memory

## APPENDIX B

### Installing SW, Configuring and Starting SUT

Before starting the SIP server on the SUT, the following tasks need to be carried out.

1. Modify the IP address of the SUT in the configuration file

- 1.1. /etc/opensips/opensips.cfg

2. Start the MySQL daemon with the command “/etc/init.d/mysql start”

#### 3. Install opensips

- 3.1. Download the opensips-1.6.2-notls [10] source code.

- 3.2. Untar and unzip the downloaded file.

- 3.3. Read the {opensips}/INSTALL file for supported architectures and installation requirements (only MySQL devel libs are needed from the optional list)

- 3.4. Edit {opensips}/Makefile: Remove mysql from the excluded modules list (exclude\_modules?=)

- 3.5. Compile opensips: make prefix={opensips} all

- 3.6. Install opensips: make prefix={opensips} install

#### 4. Populate the SIP subscriber database as shown in the below steps

4.1. Edit "{opensips}/scripts/opensipsdbctl.base" to include a new line:

```
DBENGINE=MYSQL (or uncomment it if it is already there)
```

4.2. Also uncomment the same line at {opensips}/scripts/opensipsctlrc

4.3. Run "{opensips}/scripts/opensipsdbctl create" (answer NO to all the questions about optional tables)

4.4. Configure opensips to use the MySQL database and modify the settings pertaining to this experiment based on the configuration file "{CPB}/examples/opensips\_example.cfg"

4.4.1. Edit {opensips}/etc/opensips/opensips.cfg

```
Change:  debug=0
```

```
Change:  children={USER_SPECIFIED_VALUE} (number of threads to be used)
```

```
Add:    alias=open-ims.test
```

```
Uncomment: #listen=udp: 10.20.106.105:5060 (and change the IP address)
```

```
Uncomment: #loadmodule "mysql.so"
```

```
Uncomment: #loadmodule "auth.so"
```

```
Uncomment: #loadmodule "auth_db.so"
```

```
Uncomment: #modparam("usrloc", "db_mode", 2)
```

```
Uncomment: #modparam("usrloc", "db_url", "mysql://opensips:opensipsrw@10.20.106.105/opensips-1.6.2-notls") (and change the end to @localhost/opensips)
```

Uncomment: #modparam("auth\_db", "calculate\_ha1", yes)

Uncomment: #modparam("auth\_db", "password\_column", "password")

Uncomment: #modparam("auth\_db", "db\_url",  
"mysql://opensips:opensipsrw@192.168.1.3/opensips\_1\_3") (and change the  
end to @localhost/opensips)

Uncomment: ##if (!www\_authorize("", "subscriber"))

Uncomment: ##{

Uncomment: ## www\_challenge("", "0");

Uncomment: ## exit;

Uncomment: ##}

\*: If several network interfaces are to be added, just add more lines

"listen=udp:ANOTHER\_IP-ADDRESS:5060" rows

#### 4.4.2. Add users to Opensips MySQL database

##### 4.4.2.1. Edit {opensips}/scripts/opensipsctrlc:

Uncomment: #DBENGINE=MYSQL

Uncomment: #DBRWUSER=opensips

Uncomment: #DBRWPW="opensipsrw"

##### 4.4.2.2. Edit {opensips}/scripts/opensipsctl:

Change: TEST="true"

##### 4.4.2.3. Copy {CPB}/scripts/generate\_db\_subscribers.sh to

{opensips}/scripts/ and run it

(By default the script generates 100000 SIP users. The number of SIP users to be generated can be changed by modifying the value of "LIMIT".

Beware that it might take several hours to generate all the users.

If the IMS Bench SIPp test run suddenly ends with no apparent reason, the SIP users in the database should be more than the default 100000 users)

If, for some reason, the SIP users need to be deleted from the database, copy “{CPB}/scripts/remove\_db\_subscribers.sh” to “{opensips}/scripts/” and run it (and possibly change the amount of users to be deleted)

## **5. Start the SIP Server with the following command**

5.1. `./opensips -l 10.20.106.105 -m 3200 ../etc/opensips/opensips.cfg`

## **6. Start the tool “cpum”, which measures the CPU utilization and memory usage statistics of the SUT**

6.1. `@ims_bench> ./cpum 10.20.106.106:5000` (IP address of the Manager followed by the Port number)

## APPENDIX C

### Configuring and Starting Manager

#### 1. Create IMS-Bench configuration

== IMS Benchmark Configuration - Main Menu ==

- 1) Test System Setup
- 2) SUT Setup
- 3) Traffic Time Profile
- 4) Traffic Set
- 5) Users provisioning
- 6) Options

Enter option to select OR 'q' when done > q

Do you want to generate the benchmark run files? [Y/n] > Y

Creating benchmark run files in './ims\_bench\_0'

Saving configuration to ./ims\_bench\_0/ims\_bench.xml

Copying scen/ims\_rereg.xml

Copying scen/ims\_reg.xml

Copying scen/ims\_uac.xml

Copying scen/ims\_dereg.xml

Copying scen/ims\_msgc.xml

Copying scen/ims\_uas.xml

Copying scen/ims\_msgs.xml

Generating user data files for 100000 users for TS1. This may take some time...

Generating run script for TS1....

All files necessary to execute the benchmark have been copied or created into './ims\_bench\_0' directory.



The current configuration has also been saved under  
'./ims\_bench\_0/ims\_bench.xml'.

You may later prepare a new benchmark configuration based on this one by  
running

```
./scripts/ims_bench.pl ./ims_bench_0/ims_bench.xml
```

## 2. Prepare the Test Systems (TS) for benchmark execution

2.1. Make sure all the TSs have the necessary authentication keys set up so that  
SSH sessions can be opened from this host without requiring a password to be  
entered.

2.2. Check that all TSs have the prerequisite software installed to run the IMS  
Bench SIPp (for example GSL library). See the installation instructions in the  
documentation for more details on this.

2.3. Change to './ims\_bench\_0' directory

```
cd ./ims_bench_0
```

2.4. Copy the necessary files onto all the TSs by executing

```
./prepare.sh
```

## 3. Start the Manager

3.1. Copy scenario files from the directory CPB (Control Plane Benchmark). It  
should be noted that these files are copied from CPB only at the Manager's  
side and not at the SUT and UAC. The "IMS-Bench SIPp" is used on SUT and  
UAC for different purpose than it is used on the Manager's side.

```
cp {CPB}/scenarios/* ims_bench/scen
```

3.2. Edit {IMS\_Bench@MANAGER}/ims\_bench\_0/manager.xml (in order to  
configure "manager.xml", refer to the configuration file  
{CPB}/examples/manager\_example.xml)

3.2.1. Add the following line under the "Scenario Parameters" section.

```
<param name="scenario_path" value="scen"/>
```

3.2.2. Set the “cps” value under the “Pre-Registration Phase” to the maximum, the SUT can handle (probably something like 50 or more)

3.2.3. Set the following values under the “Benchmark Run Phase”. These values can be modified based on the need of the experiment accordingly.

```
<run cps="50" duration="900" step_increase="10" num_steps="9999"  
distribution="poisson" stats="2000">
```

3.3. It should be noted that the phrase "Service-Route:" should be removed from the files “ims\_reg.xml” and “ims\_rereg.xml”. These files are located in the directories "scen" and "ims\_bench\_0".

3.4. Start the Manager at the level of the directory "ims\_bench" but not at the level of the directory "ims\_bench\_0".

```
./manager -f ims_bench_0/manager.xml
```

## APPENDIX D

### PREREQUISITE

The following SW is installed and the configurations are done as mentioned below prior to starting the experiment.

1. The Virtualization Technology (VT) support at HW level was turned on during the boot phase of the physical host.
2. Installed the RedHat Enterprise Linux 5.5 (RHEL5.5) on all the physical hosts. The RHEL5.5 is the OS installed on all the physical hosts in the native environment. The VMs are installed with RHEL5.5 in the virtual environment.
3. Installed the following common SW on SUT, Manager and UAC
  1. GSL library [11]
  2. Perl XML::Simple module [11]
  3. Gnuplot 4.2 [11]
  4. ntpd [11]
  5. ptpd [11]
  6. IMS-Bench SIPp [11]
4. Installed the following SW on SUT, which is specific to SUT
  1. MySQL database
  2. opensips-1.6.2-notls [10]
5. Installed the following SW on Manager, which is specific to Manager
  1. Control Plane Bench Mark [9]. This is used as the reference configuration in this experiment to modify the default configuration of the "IMS-Bench SIPp [11]". The detailed instructions to perform these configuration changes are described, while generating the IMS-Bench Mark configuration at the Manager.
6. OS (RHEL5.5) Configuration

The capacity of the physical hosts was downgraded by modifying the configuration file "grub.conf", which is located in the directory "/boot/grub" as follows

1. Edit the file "/boot/grub/grub.conf" and set the values "maxcpus=1" and "mem=6G", which means that only 1 CPU is used and 6GB of RAM is used out of all the resources available in the host.
2. The physical hosts, which were deployed in the experiments had 8 CPUs and 16 GB of RAM.
3. Turn off "kdump" with the following command
  1. `chkconfig --list | grep :on`
  2. `chkconfig kdump`

Turn off "swap" with the command "swapoff -av"

## APPENDIX E

### RESULTS OF THE EXPERIMENT IN VIRTUAL ENVIRONMENT

#### Test Report (localhost.localdomain)

### Summary

This report shows the result of a benchmark run performed by "IMS Bench SIPp", an implementation of the **IMS/NGN Performance Benchmark** suite, **ETSI TS 186.008**.

The test was started on 22-Apr-2010 01:41, and the total time for the test execution was **2h 10m 47s**. The **Design Objective Capacity (DOC)** is **100** scenarios per second.

The following systems and parameters were used for the test. The full list of IMS benchmark parameters can be found in Appendix.

Role	Server	IP	Nb Users
SUT 1	localhost.localdomain	10.20.106.105	
Manager	localhost.localdomain	127.0.0.1	
TS1	localhost.localdomain	10.20.106.107	10000
TS2	localhost.localdomain	10.20.106.20	10000
TS3	localhost.localdomain	10.20.106.21	10000
TS4	localhost.localdomain	10.20.106.22	10000
TS5	localhost.localdomain	10.20.106.23	10000
TS6	localhost.localdomain	10.20.106.24	10000
TS7	localhost.localdomain	10.20.106.25	10000
TS8	localhost.localdomain	10.20.106.26	10000
TS9	localhost.localdomain	10.20.106.27	10000
TS10	localhost.localdomain	10.20.106.28	10000

Parameter Name	Parameter Value	Parameter Info
RingTime	5000	Ringing Time (ms)
HoldTime	30000	Conversation Time (ms)
RegistrationExpire	1000000	Registration Timeout (ms)
TransientTime	30	Time after the start of a step for which data is ignored (in seconds)

The following table shows the average of the key measurements for each step of the test. Each step is characterized by the requested load, the effective load, the global IHS (total of all

Inadequately handled scenarios for this step divided by number of Session Attempts for this step) the scenario IHS (number of inadequately handled scenarios for this step divided by thenumber of scenario attempts for this step), the CPU utilization and the available Memory on the SUT. The available Memory is expressed in MegaBytes, and the requested and effective loads in Scenarios Attempts Per Seconds (SAPS).

Note that the IHS percentages represented in this table are the number of failures for a step divided by the number of scenario attempts for this step, and so is not the average of (IHS per seconds)

	Pre-registration	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
<b>Requested load</b>	50	50	60	70	80	90	100	110
<b>Effective Load</b>	50.00	50.03	60.09	69.67	80.12	89.87	100.44	109.00
<b>Ratio ims_rereg %</b>	0.00	14.89	15.12	15.05	15.20	14.98	15.06	14.94
<b>Ratio ims_reg %</b>	100.00	2.52	2.46	2.38	2.42	2.47	2.49	2.42
<b>Ratio ims_uac %</b>	0.00	49.81	50.12	49.75	49.94	50.21	49.85	49.41
<b>Ratio ims_dereg %</b>	0.00	2.46	2.36	2.54	2.57	2.38	2.55	2.58
<b>Ratio ims_msgc %</b>	0.00	30.32	29.93	30.27	29.87	29.96	30.06	30.66
<b>CPU localhost.localdomain</b>	13.57	40.63	50.49	60.10	70.75	81.59	95.90	99.99
<b>Memory localhost.localdomain</b>	3776.43	3755.47	3735.19	3710.63	3682.42	3649.50	3613.57	3571.89
<b>SIPP CPU localhost.localdomain</b>	0.55	0.66	0.73	0.82	0.99	1.18	1.33	1.72
<b>SIPP MEM localhost.localdomain</b>	13977.39	13958.20	13957.31	13953.85	13949.01	13943.69	13937.41	13927.22
<b>IHS ims_rereg %</b>	0.00	0.00	0.00	0.00	0.00	0.00	3.49	21.25
<b>IHS ims_reg %</b>	0.00	0.00	0.00	0.00	0.00	0.00	1.75	13.57
<b>IHS ims_uac %</b>	0.00	0.00	0.00	0.00	0.00	0.19	7.10	43.16
<b>IHS ims_dereg %</b>	0.00	0.00	0.00	0.00	0.00	0.00	1.75	13.93
<b>IHS ims_msgc %</b>	0.00	0.00	0.00	0.00	0.00	0.00	4.83	29.67
<b>global IHS %</b>	0.00	0.00	0.00	0.00	0.00	0.09	5.60	34.28

The following chapters show details on different measurement, like delay between two messages, response time or number of messages per seconds.

Each measurement can be represented in one of the four following forms.

1. Evolution in function of the time. On such graphs, the raw information is plotted, like number of messages per seconds, or response time of each scenario. This graph is useful in giving for instance a good idea on the distribution of response times, and it's evolution over the time.
2. Evolution (mean) in function of the time. While previous graph gives a good indication, it may sometimes be easier to see the evolution of the mean of the measurement over a second in function of the time.
3. Histogram. This graph shows the histogram of the measurement, so how many times each value of the measurement occurred.
4. Probability. This graph gives the probability of the measurement to be higher than a certain value. This graph can be used to determine percentile for instances.

For some graphs, a cubic Bezier curve is plotted as well.

## 1. Scenario Attempts Per Second

This graph represents the number of scenario per seconds generated by the test system. For each step, the generation was based on a Poisson.

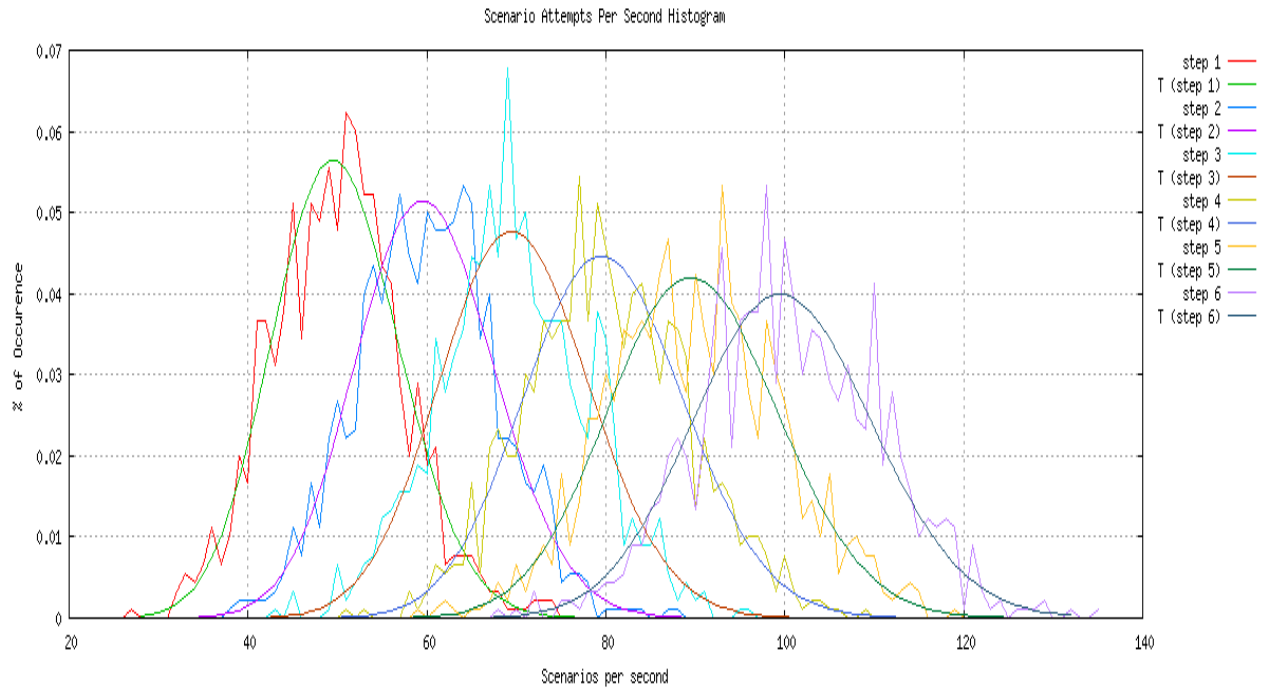
Step	Requested Load	Effective Load								
		Mean	Variance	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
Pre-reg	50	50.00	5.27	2.29	44.00	56.00	50.0	53.0	55.0	56.0
1	50	50.03	53.11	7.29	27.00	74.00	50.0	59.0	62.0	69.0
2	60	60.09	60.85	7.80	38.00	88.00	60.0	70.0	73.0	78.0
3	70	69.67	69.48	8.34	43.00	96.00	69.0	80.0	84.0	89.0
4	80	80.12	81.59	9.03	51.00	109.00	80.0	92.0	96.0	101.0
5	90	89.87	95.37	9.77	59.00	119.00	90.0	103.0	107.0	113.0
6	100	100.44	102.26	10.11	68.00	135.00	100.0	113.0	117.0	124.0
7	110	109.00	119.02	10.91	76.00	152.00	109.0	123.0	128.0	136.0

### 1.1 Scenario Attempts Per Second (Mean per second)



### 1.2 Scenario Attempts Per Second Histogram

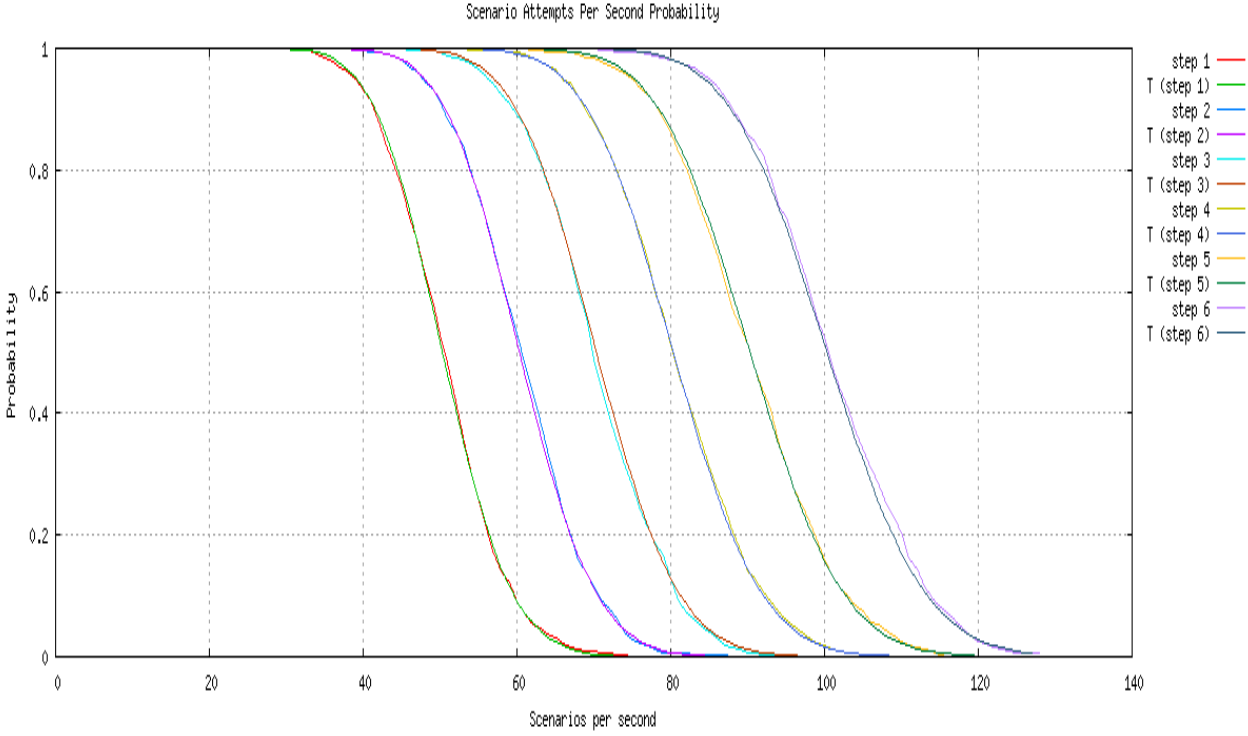
This graph shows the Histogram of the SAPS for each step. It should follow Poisson distributions.



### 1.3 Scenario Attempts Per Second Probability

This graph shows the probability distribution of the SAPS for each step. It shows the probability that the effective load is higher than x.



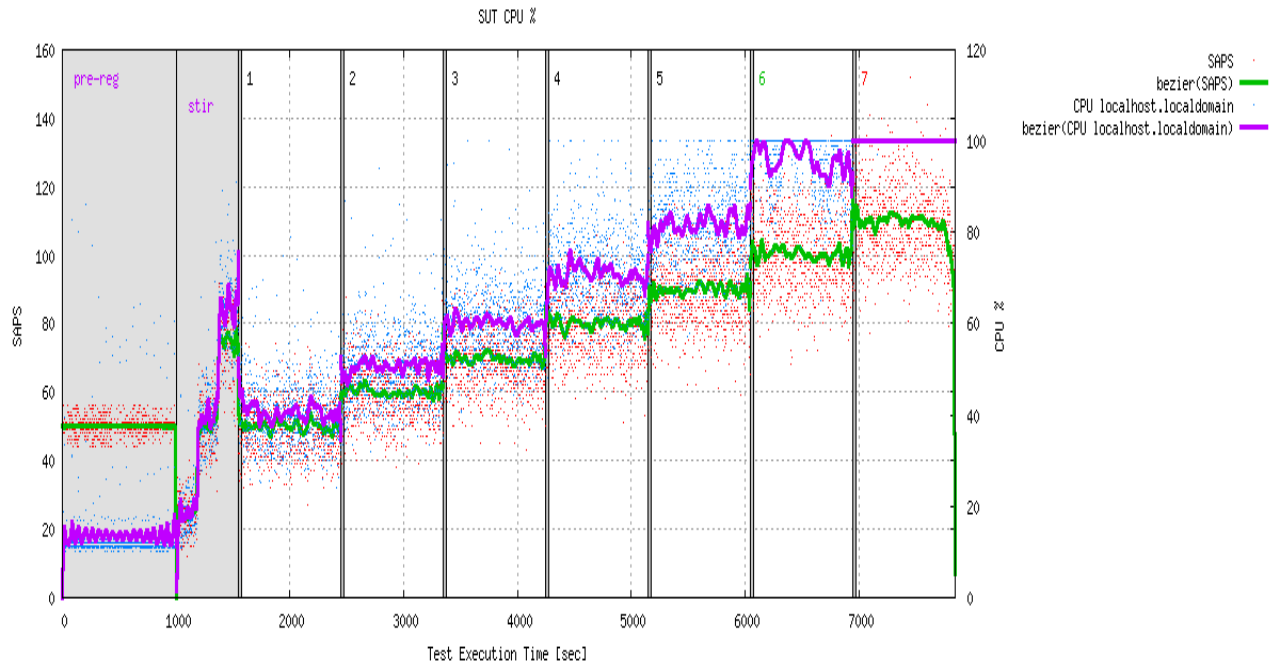


## 2 SUT CPU %

This graph represents the CPU of the system under test (SUT).

		CPU localhost.localdomain			
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum
Pre-reg	50	13.57	8.52	8.00	86.14
1	50	40.63	7.04	24.00	79.00
2	60	50.49	7.75	32.00	94.06
3	70	60.10	7.99	40.59	100.00
4	80	70.75	8.89	48.00	100.00
5	90	81.59	8.91	56.57	100.00
6	100	95.90	6.14	65.00	100.00
7	110	99.99	0.23	93.94	100.00

### 2.1 SUT CPU % over time

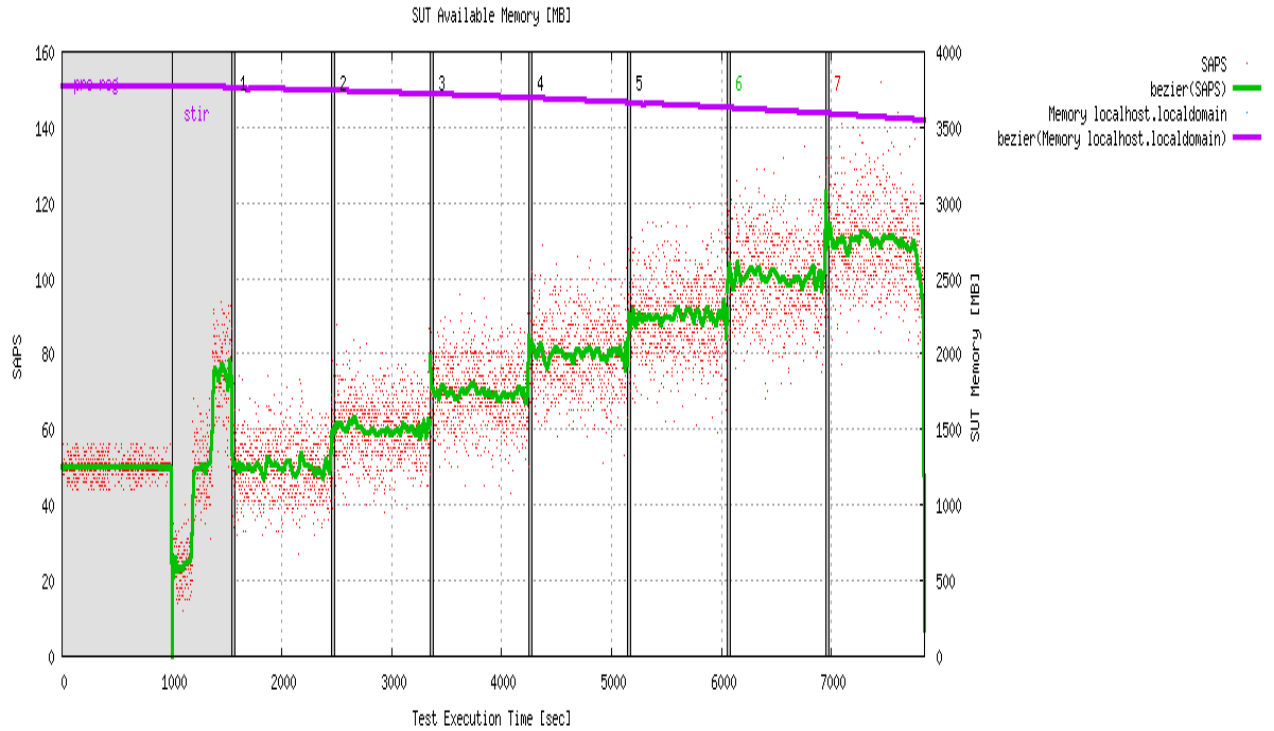


### 3 SUT Available Memory [MB]

This graph represents the Available memory on the system under test, in MBytes (SUT).

		Memory localhost.localdomain			
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum
Pre-reg	50	3776.43	0.07	3776.06	3776.60
1	50	3755.47	5.55	3745.46	3765.44
2	60	3735.19	6.64	3724.00	3746.34
3	70	3710.63	7.59	3697.80	3724.02
4	80	3682.42	8.78	3667.42	3697.70
5	90	3649.50	9.61	3632.94	3667.32
6	100	3613.57	11.02	3594.50	3632.97
7	110	3571.89	13.01	3549.86	3594.53

### 3.1 SUT Available Memory [MB] over time

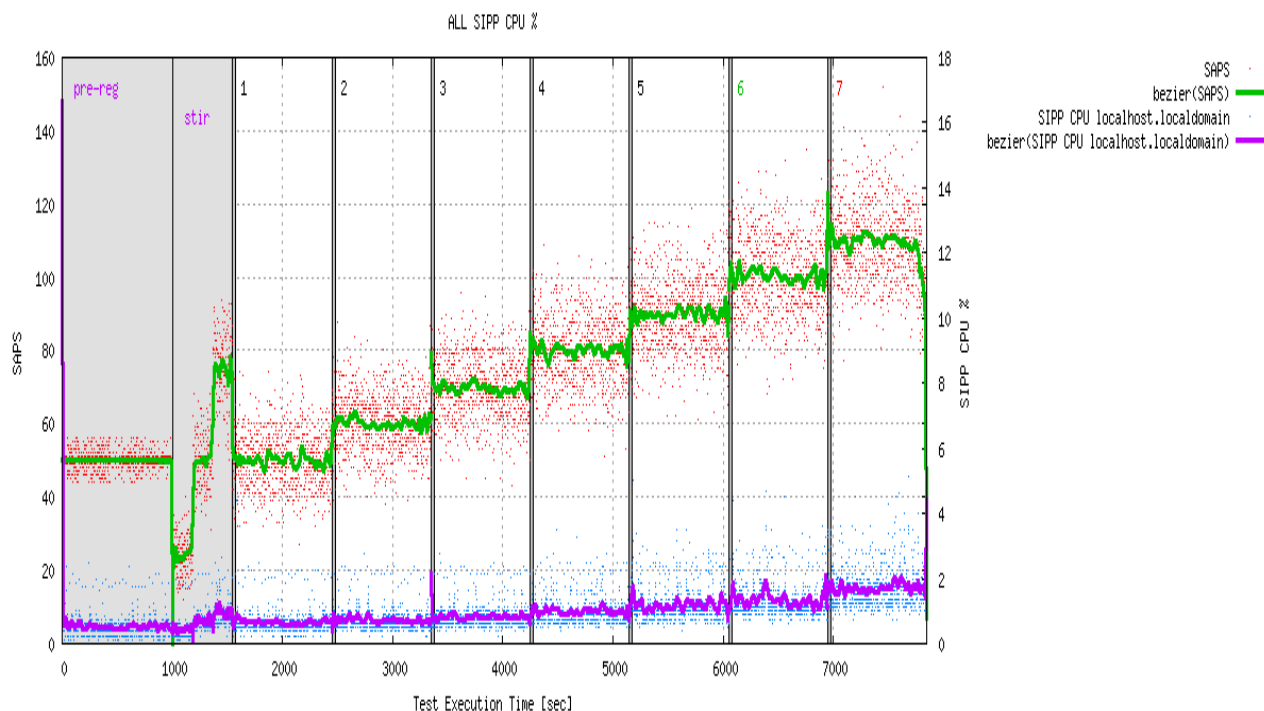


### 4 ALL SIPP CPU %

This graph represents the CPU of SIPP on ALL Test Machines

SIPP CPU localhost.localdomain					
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum
Pre-reg	50	0.55	0.61	0.13	16.65
1	50	0.66	0.36	0.25	4.39
2	60	0.73	0.33	0.25	2.77
3	70	0.82	0.34	0.25	2.75
4	80	0.99	0.40	0.38	3.51
5	90	1.18	0.50	0.38	5.03
6	100	1.33	0.53	0.63	4.67
7	110	1.72	0.59	0.75	5.14

## 4.1 ALL SIPP CPU % over time

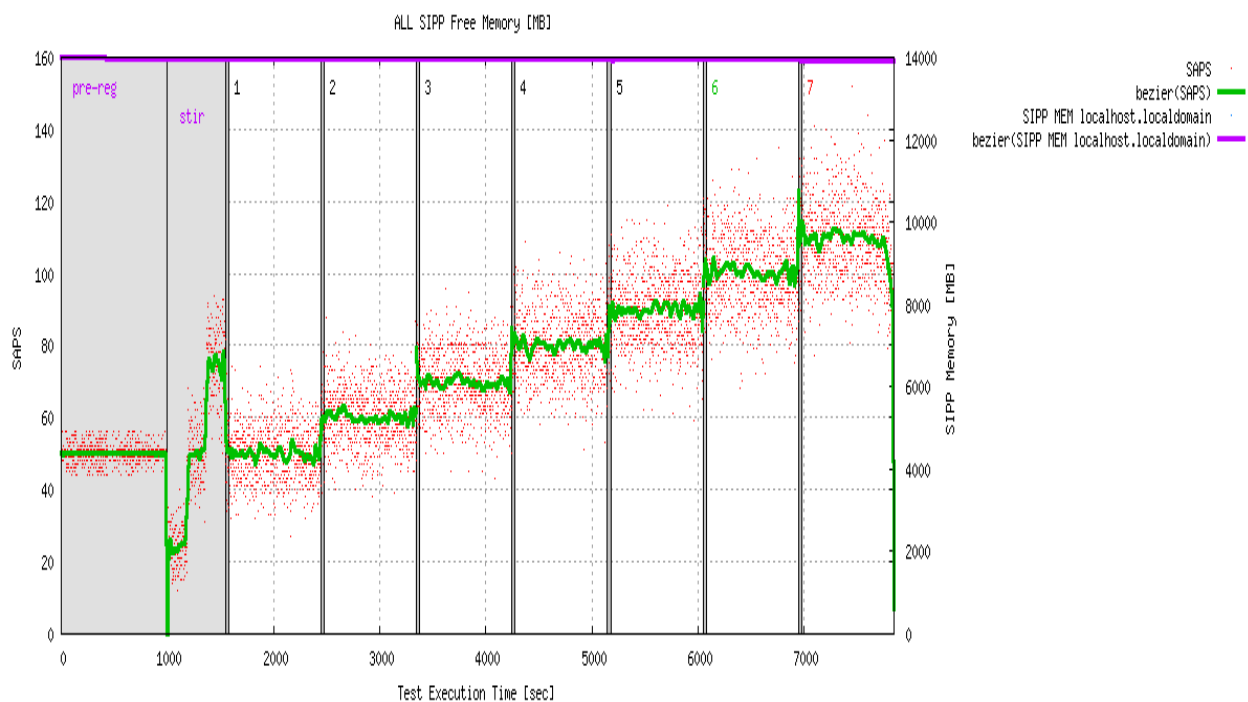


## 5 ALL SIPP Free Memory [MB]

This graph represents the free memory of SIPP on ALL Test Machines, in Mbytes

		SIPP MEM localhost.localdomain			
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum
Pre-reg	50	13977.39	3.43	13971.39	13986.89
1	50	13958.20	2.79	13943.98	13959.85
2	60	13957.31	0.80	13956.13	13959.35
3	70	13953.85	1.37	13949.68	13956.51
4	80	13949.01	1.38	13946.71	13952.53
5	90	13943.69	2.89	13930.96	13947.70
6	100	13937.41	1.82	13934.55	13941.50
7	110	13927.22	3.81	13920.42	13934.56

## 5.1 ALL SIPP Free Memory [MB] over time

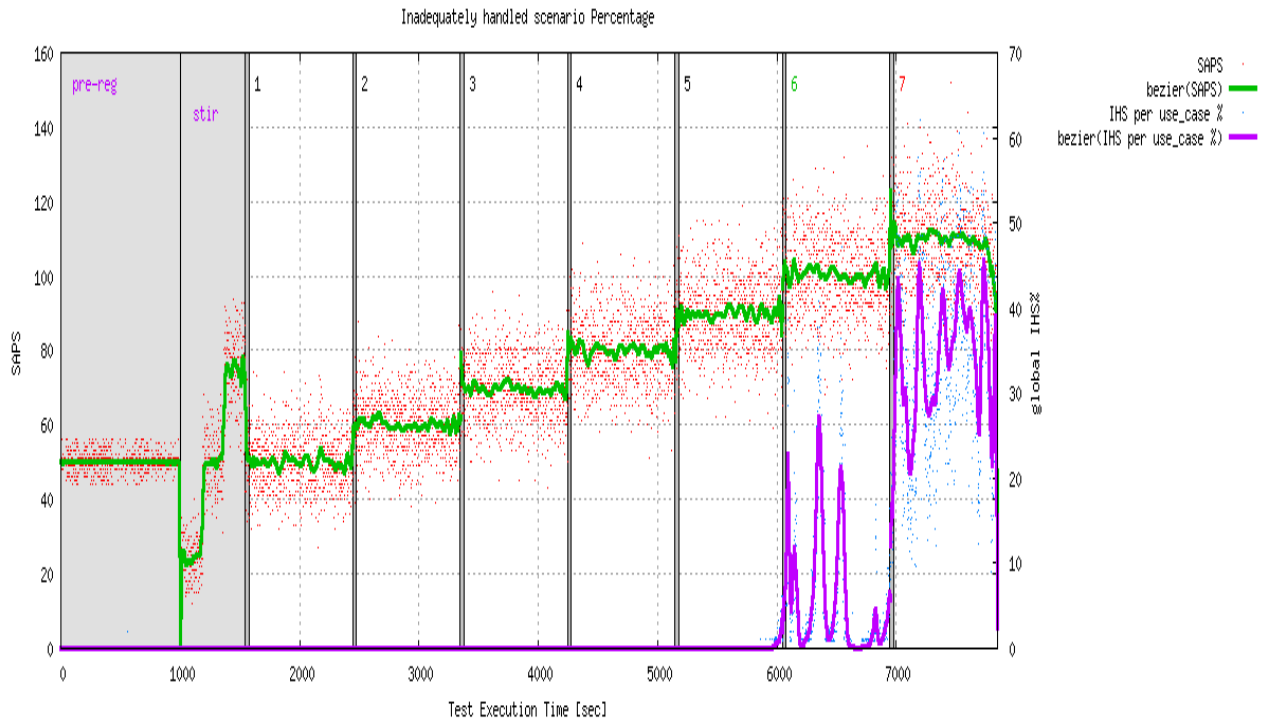


## 6 Inadequately handled scenario Percentage

This graph represents the percentage of inadequately handled scenarios.

Step	Requested Load	IHS per use_case %			
		Mean	Standard Deviation	Minimum	Maximum
Pre-reg	50	0.00	0.06	0.00	2.00
1	50	0.00	0.00	0.00	0.00
2	60	0.00	0.00	0.00	0.00
3	70	0.00	0.00	0.00	0.00
4	80	0.00	0.00	0.00	0.00
5	90	0.09	0.49	0.00	6.25
6	100	5.58	8.73	0.00	47.19
7	110	33.60	11.01	5.61	62.14

### 6.1 Inadequately handled scenario Percentage over time

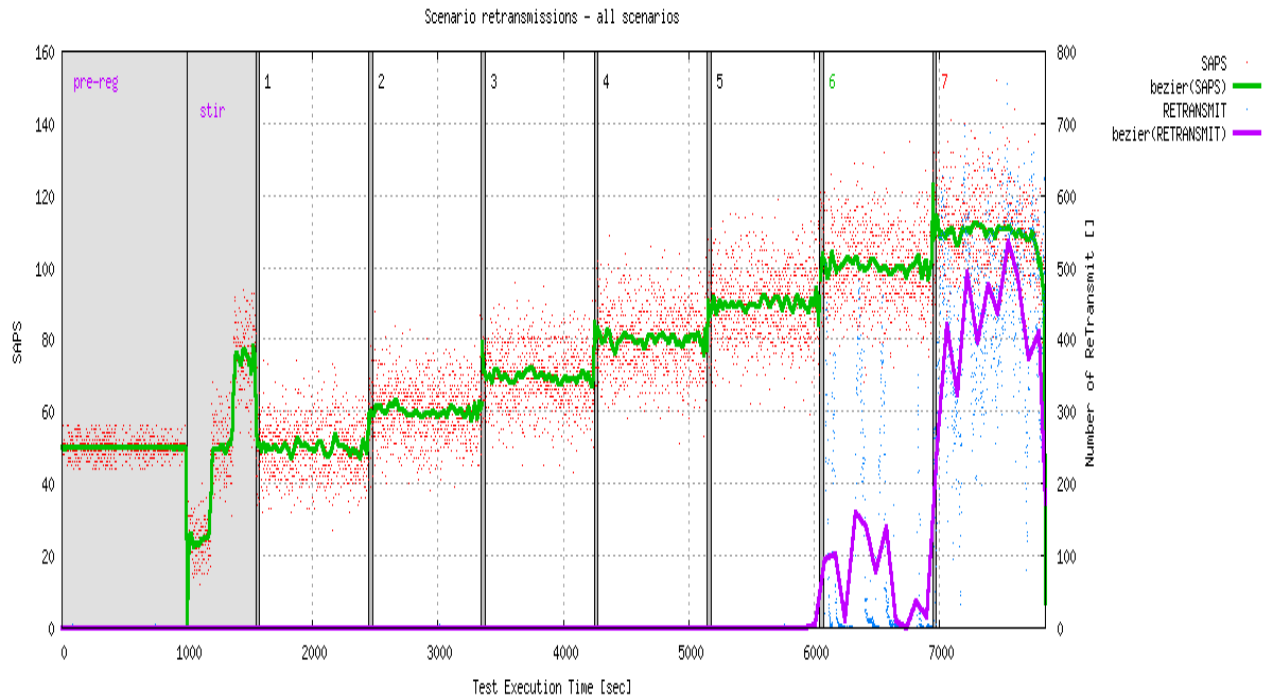


### 7 Scenario retransmissions - all scenarios

This graph represents the number of retransmissions per seconds for all scenarios.

		RETRANSMIT							
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
Pre-reg	50	0.03	0.35	0.00	6.00	0.0	0.0	0.0	0.0
1	50	0.11	0.35	0.00	3.00	0.0	0.0	1.0	2.0
2	60	0.13	0.41	0.00	5.00	0.0	1.0	1.0	2.0
3	70	0.20	0.46	0.00	3.00	0.0	1.0	1.0	2.0
4	80	0.23	0.52	0.00	3.00	0.0	1.0	1.0	2.0
5	90	0.24	0.53	0.00	4.00	0.0	1.0	1.0	2.0
6	100	71.05	131.29	0.00	505.00	3.0	324.0	378.0	462.0
7	110	428.47	129.40	0.00	755.00	442.0	578.0	619.0	654.0

## 7.1 Scenario retransmissions - all scenarios over time



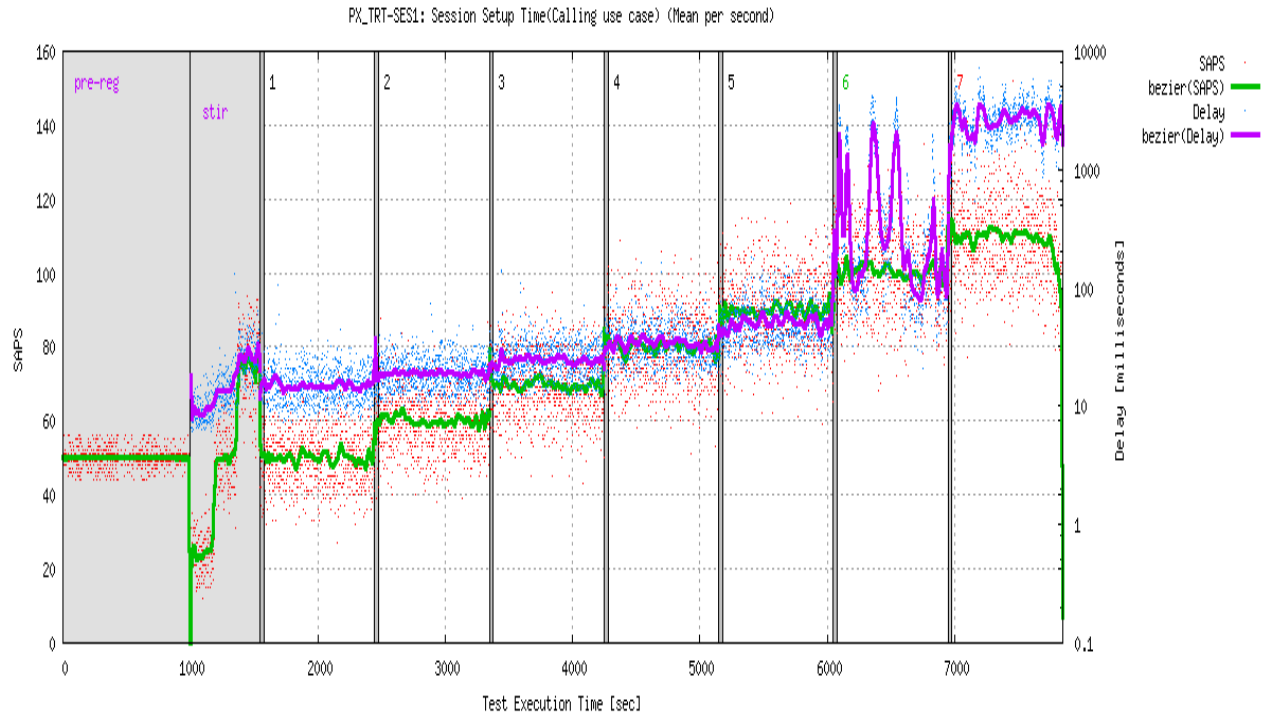
## 7 Calling

### 7.1 PX\_TRT-SES1: Session Setup Time

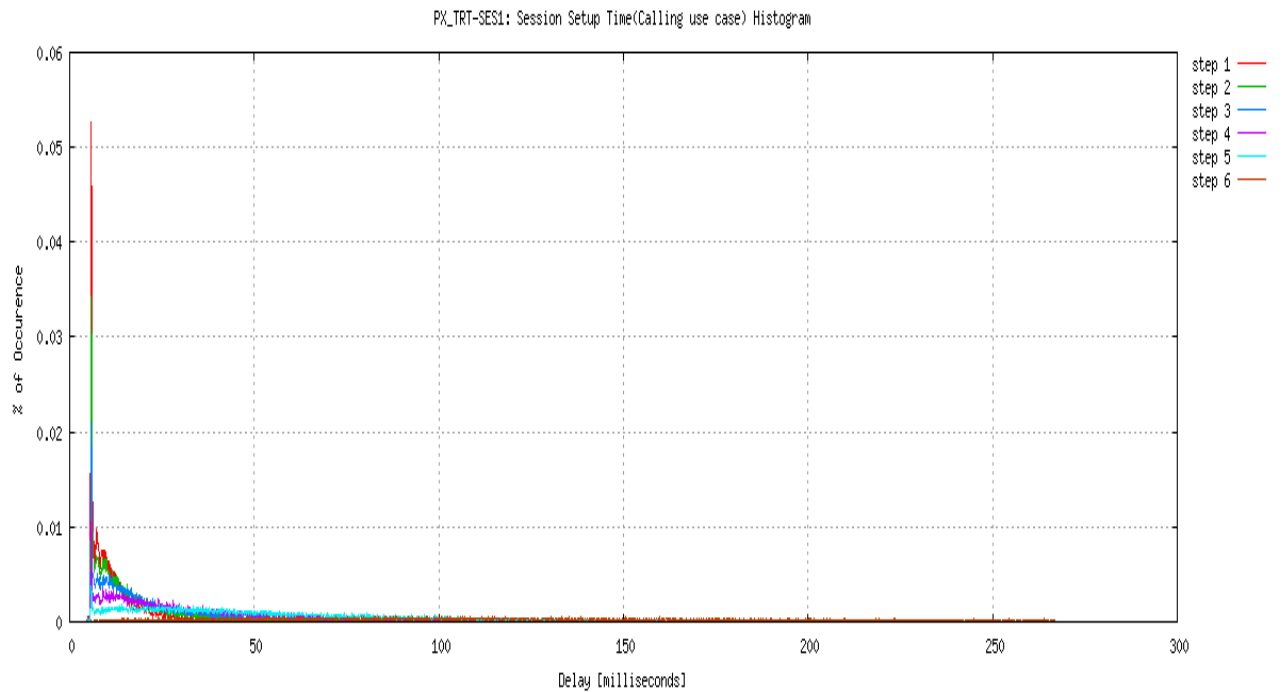
This graph represents the delay between the Caller sending INVITE and callee receiving ACK.

		PX_TRT-SES1 (msec)							
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
1	50	15.99	17.13	4.67	297.58	10.1	32.9	53.8	84.5
2	60	20.17	19.42	4.97	240.38	13.1	44.4	64.6	95.1
3	70	26.36	25.04	4.58	408.30	17.5	59.2	79.0	112.0
4	80	37.12	31.59	4.75	341.53	26.9	80.8	100.5	141.5
5	90	56.51	64.90	4.44	7986.56	45.5	114.0	138.3	185.7
6	100	596.97	1377.44	0.00	28308.66	190.2	1371.0	2433.0	6103.0
7	110	2708.71	3479.56	0.00	35815.60	1395.0	5814.0	8867.0	17701.0

### 7.1.1 PX\_TRT-SES1: Session Setup Time(Calling use case) (Mean per second)



### 7.1.2 PX\_TRT-SES1: Session Setup Time(Calling use case) Histogram



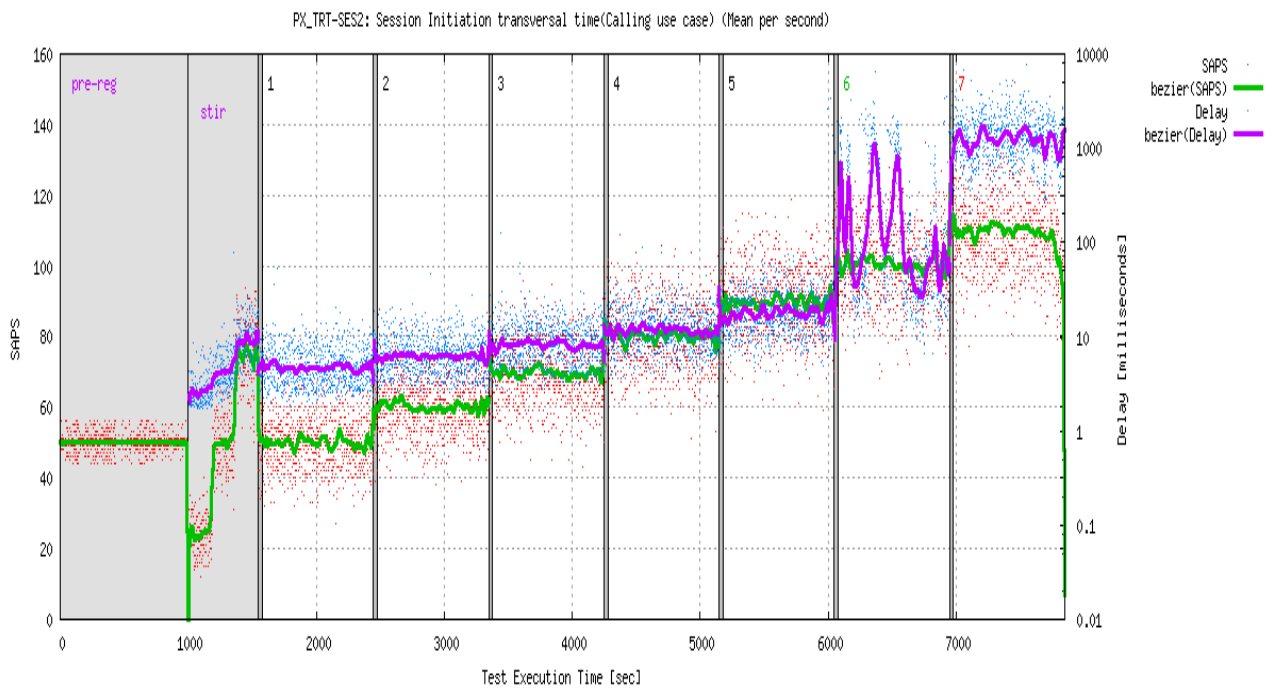


## 7.2 PX\_TRT-SES2: Session Initiation transversal time

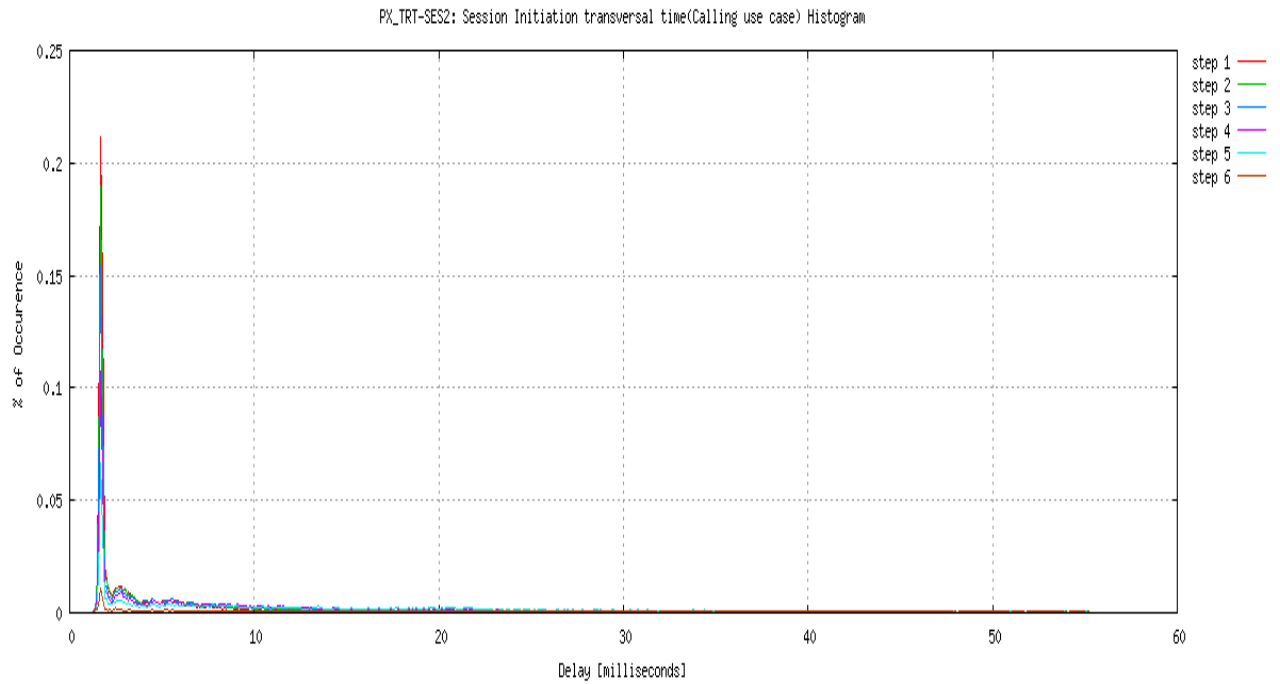
This graph represents the delay between the caller sending INVITE and the callee receiving INVITE.

Step	Requested Load	PX_TRT-SES2 (msec)							
		Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
1	50	5.37	9.61	1.32	246.27	1.8	10.2	18.9	55.1
2	60	6.97	11.08	1.26	188.71	2.7	15.5	27.4	62.4
3	70	9.46	14.10	1.22	302.29	4.1	23.1	37.6	68.6
4	80	13.53	16.67	1.25	213.38	7.0	35.4	49.0	75.6
5	90	28.25	950.73	1.13	138902.46	13.5	49.9	63.3	86.0
6	100	331.28	2930.34	1.21	180812.13	66.5	341.3	820.2	3824.0
7	110	1416.93	5282.25	1.61	200797.60	329.3	1863.0	3856.0	15896.0

### 7.2.1 PX\_TRT-SES2: Session Initiation transversal time(Calling use case) (Mean per second)



### 7.2.2 PX\_TRT-SES2: Session Initiation transversal time(Calling use case) Histogram

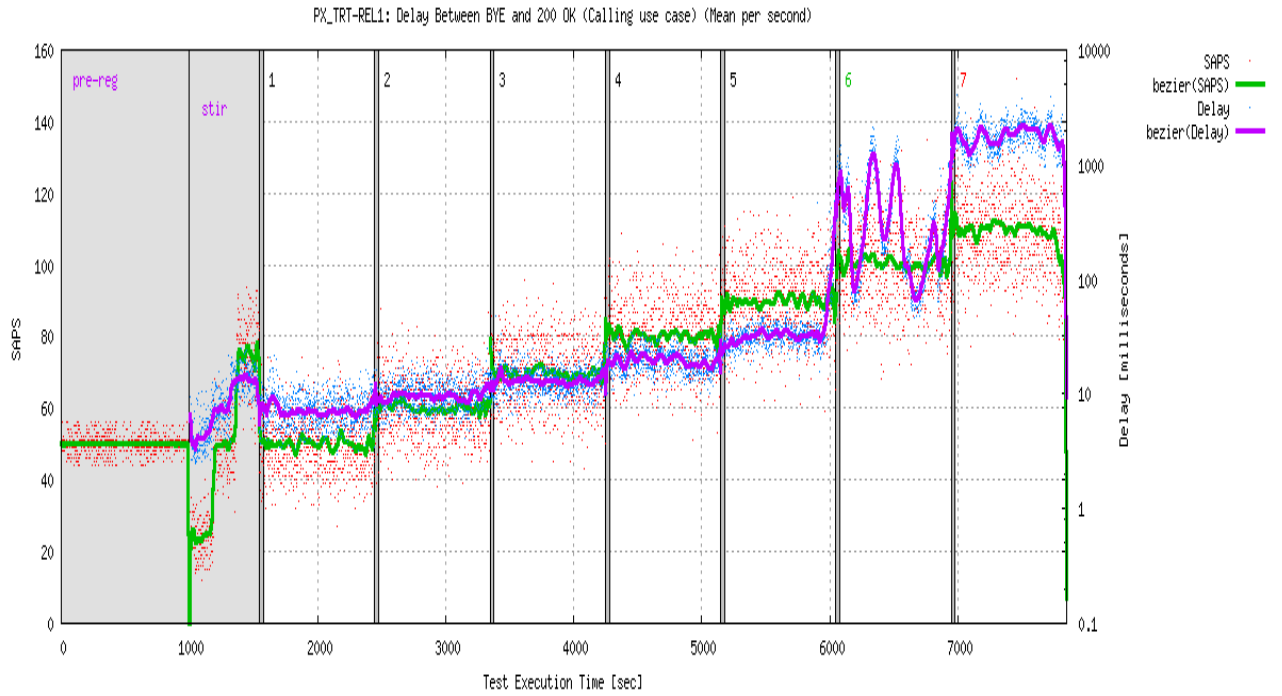


### 7.3 PX\_TRT-REL1: Delay Between BYE and 200 OK

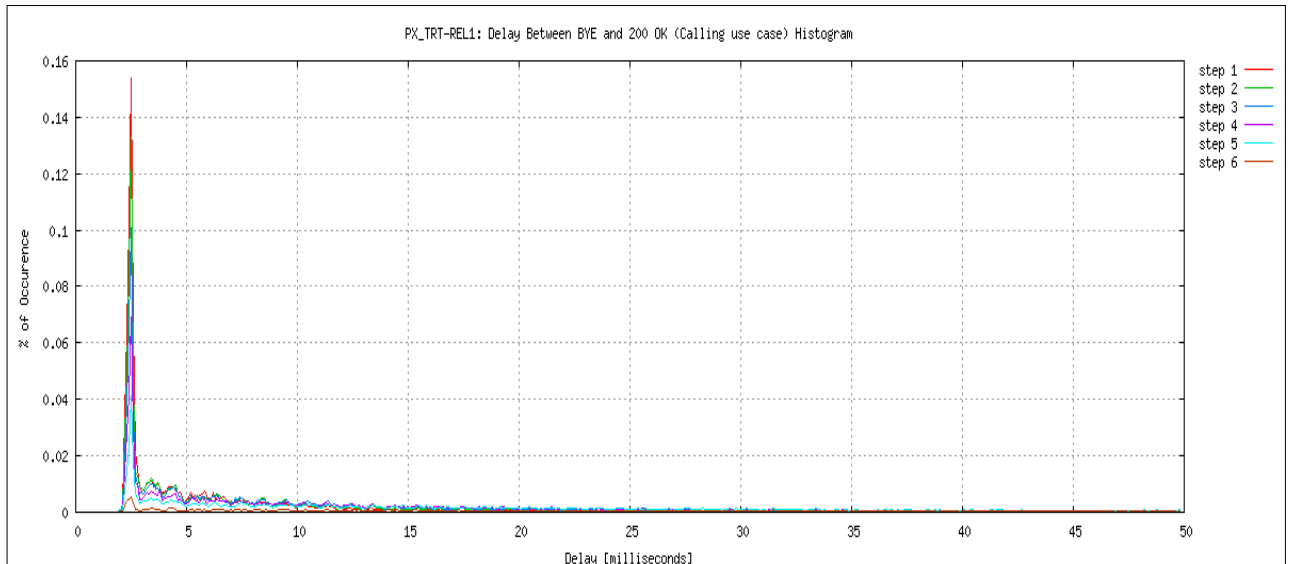
This graph represents the delay between the first BYE and the corresponding 200 OK.

		PX_TRT-REL1 (msec)							
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
1	50	7.54	12.28	1.99	301.28	3.3	14.3	26.6	67.9
2	60	9.90	15.11	1.94	405.43	4.3	21.5	38.6	79.3
3	70	13.42	20.14	1.94	398.60	6.2	32.1	54.2	92.5
4	80	19.95	25.29	1.87	338.41	10.1	52.8	76.8	112.2
5	90	44.08	215.98	1.76	19962.70	19.8	89.7	117.4	243.4
6	100	412.88	1078.27	1.82	28301.80	124.1	964.1	1687.0	4202.0
7	110	1794.55	2791.64	2.19	31041.95	958.4	3968.0	6338.0	16025.0

### 7.3.1 PX\_TRT-REL1: Delay Between BYE and 200 OK (Calling use case) (Mean per second)



### 7.3.2 PX\_TRT-REL1: Delay Between BYE and 200 OK (Calling use case) Histogram



### 7.4 PX\_TRT-SES3: INVITE and re-INVITE cost

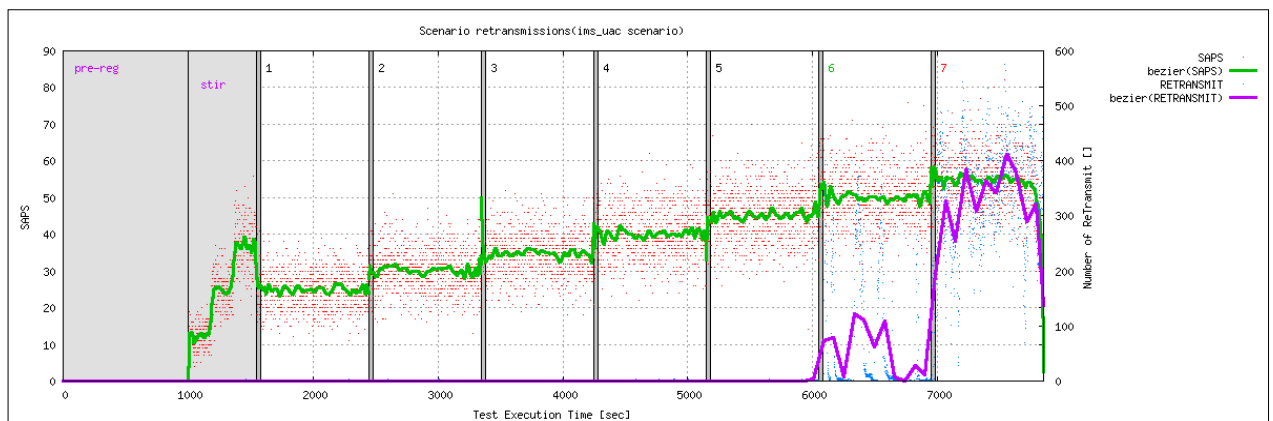
This graph represents the caller sending first INVITE and callee receiving second ACK.

### 7.5 ims\_uac : Scenario retransmissions

This graph represents the number of retransmissions per seconds for this scenario.

		RETRANSMIT							
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
Pre-reg	50	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0
1	50	0.10	0.33	0.00	3.00	0.0	0.0	1.0	1.0
2	60	0.11	0.38	0.00	4.00	0.0	0.0	1.0	2.0
3	70	0.18	0.43	0.00	3.00	0.0	1.0	1.0	2.0
4	80	0.20	0.49	0.00	3.00	0.0	1.0	1.0	2.0
5	90	0.20	0.48	0.00	3.00	0.0	1.0	1.0	2.0
6	100	55.24	101.36	0.00	394.00	3.0	251.0	292.0	356.0
7	110	331.82	98.21	0.00	576.00	343.0	445.0	472.0	507.0

#### 7.5.1 Scenario retransmissions(ims\_uac scenario) over time

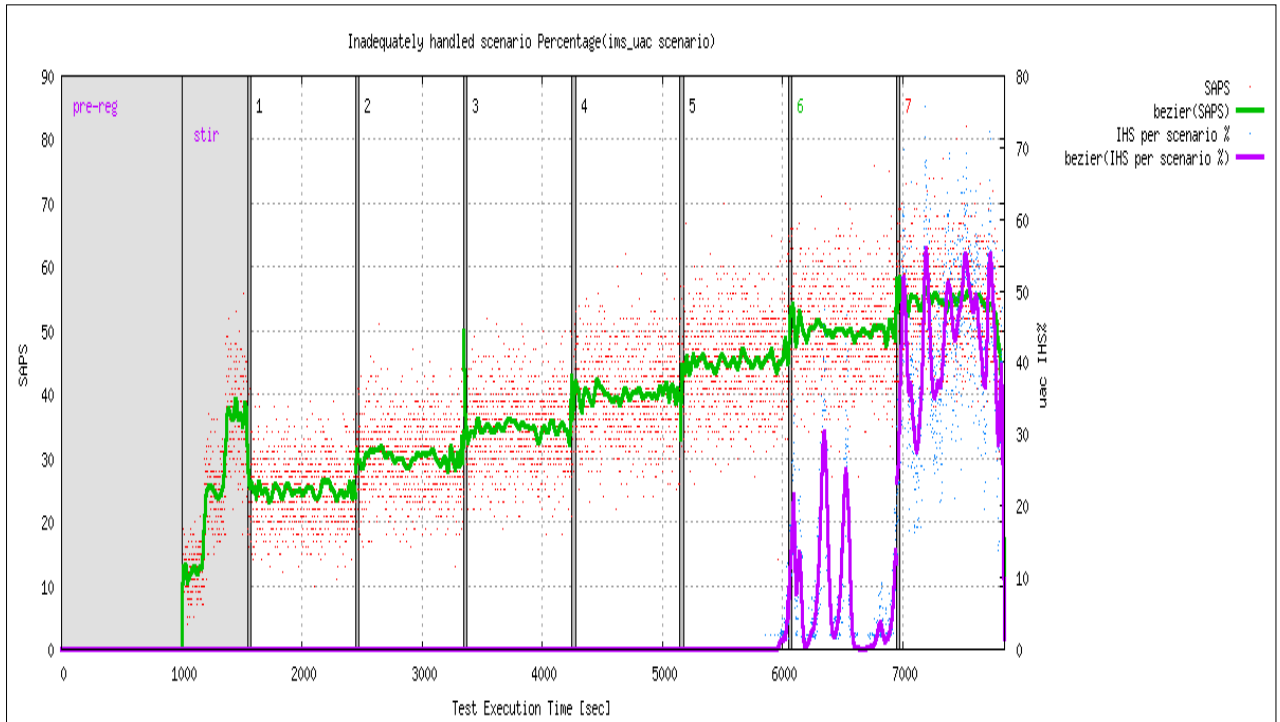


### 7.6 ims\_uac : Inadequately handled scenario Percentage

This graph represents the percentage of Inadequately handled scenarios for the uac.

		IHS per scenario %			
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum
Pre-reg	50	0.00	0.00	0.00	0.00
1	50	0.00	0.00	0.00	0.00
2	60	0.00	0.00	0.00	0.00
3	70	0.00	0.00	0.00	0.00
4	80	0.00	0.00	0.00	0.00
5	90	0.18	0.92	0.00	12.12
6	100	7.16	9.78	0.00	47.37
7	110	42.19	11.92	3.12	75.81

### 7.6.1 Inadequately handled scenario Percentage(ims\_uac scenario) over time



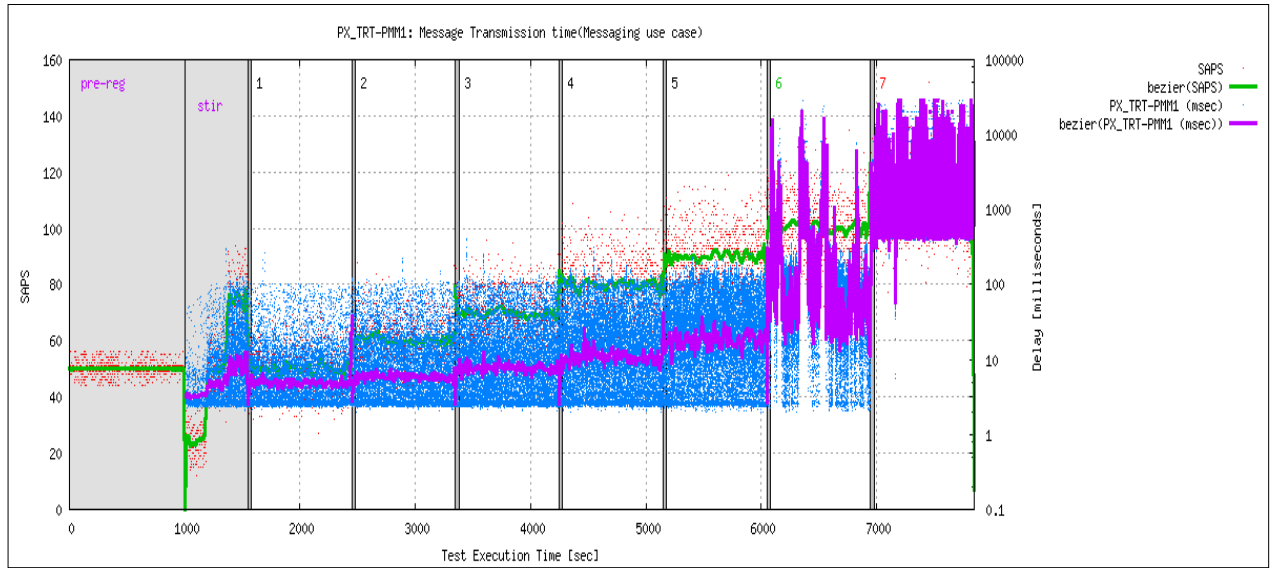
## 8 Messaging

### 8.1 PX\_TRT-PMM1: Message Transmission time

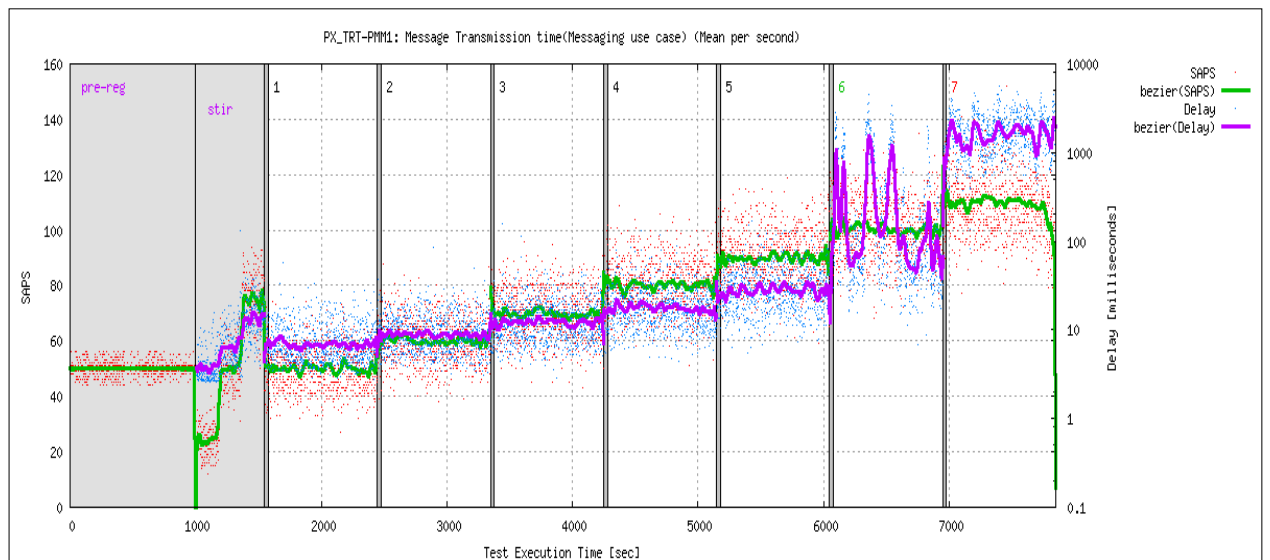
This graph represents the delay between the message and the 200 OK.

Step	Requested Load	PX_TRT-PMM1 (msec)							
		Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
1	50	7.85	12.64	2.12	270.26	3.3	15.2	28.7	70.9
2	60	10.07	15.03	2.02	220.94	4.3	21.8	38.8	81.6
3	70	14.25	21.14	2.05	416.83	6.3	35.0	58.9	98.5
4	80	20.48	25.95	1.99	314.07	10.2	54.1	77.8	117.0
5	90	32.90	34.47	1.96	292.93	20.3	84.8	106.8	143.9
6	100	378.99	991.85	1.99	28042.74	119.0	726.2	1330.0	4191.0
7	110	1722.02	2654.25	2.59	29831.42	935.4	3959.0	5064.0	15967.0

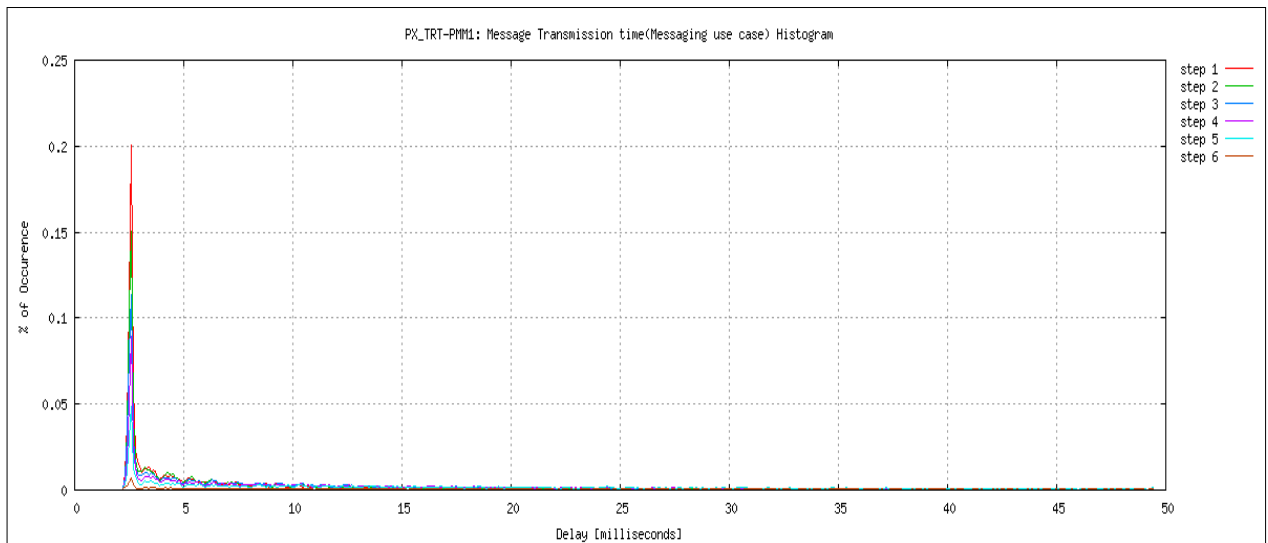
### 8.1.1 PX\_TRT-PMM1: Message Transmission time(Messaging use case) over time



### 8.1.2 PX\_TRT-PMM1: Message Transmission time(Messaging use case) (Mean per second)



### 8.1.3 PX\_TRT-PMM1: Message Transmission time(Messaging use case) Histogram



### 8.2 PX\_TRT-PMM2: Message Transmission time (error case)

This graph represents the delay between the message and the 404 Not Found.

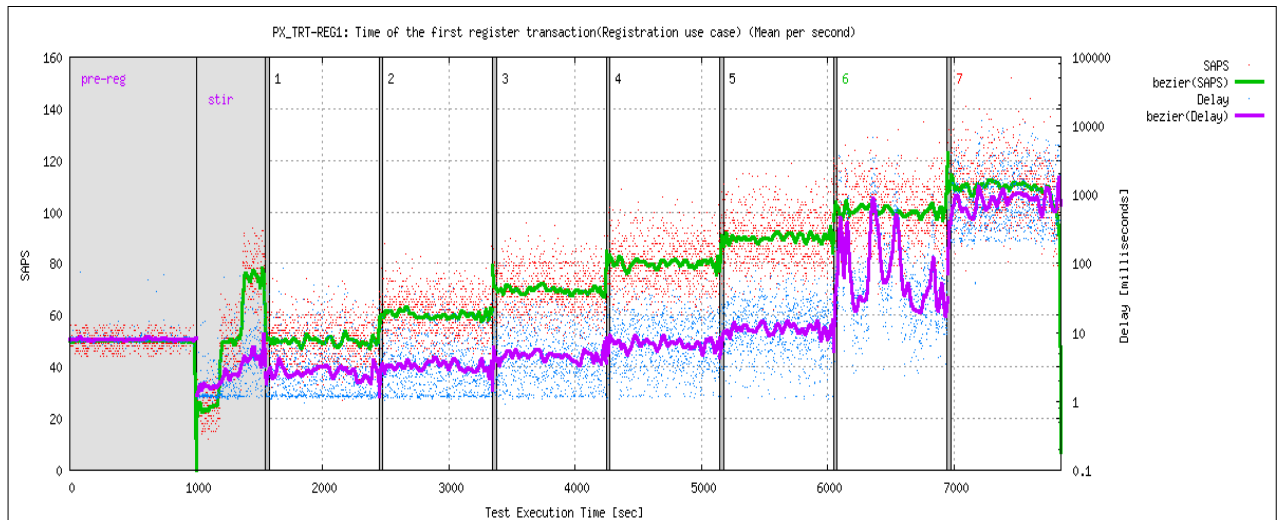
## 9 Registration

### 9.1 PX\_TRT-REG1: Time of the first register transaction

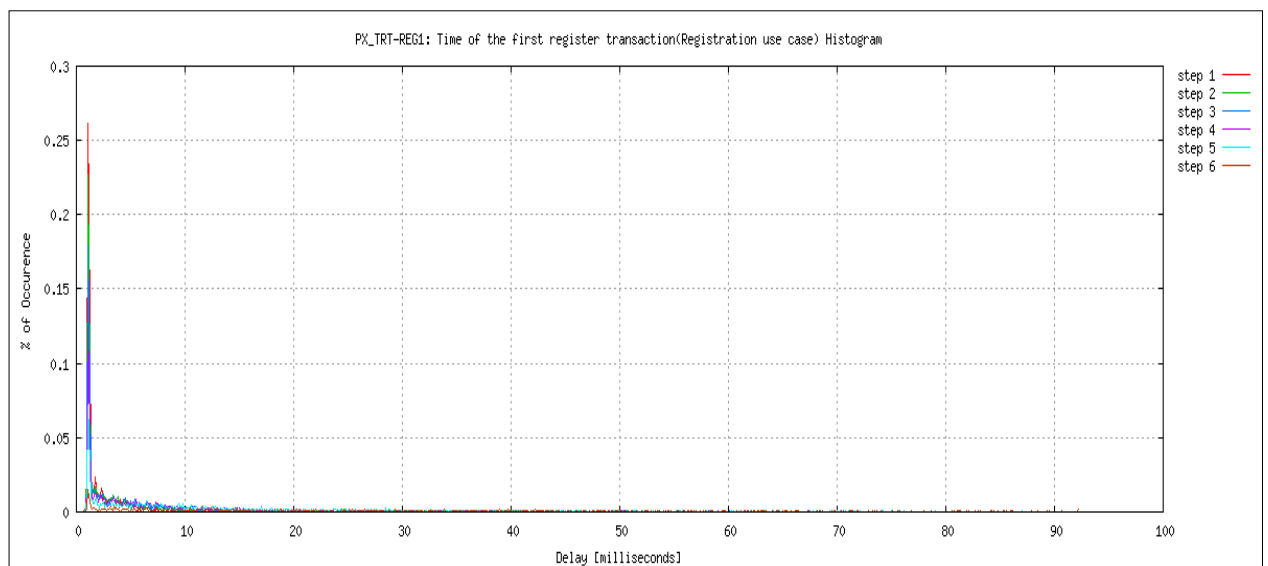
This graph represents the time of the first register transaction in the registration use\_cases i.e. the time between the REGISTER and the 401 Unauthorized for all scenarios in the Registration use\_case.

Step	Requested Load	PX_TRT-REG1 (msec)							
		Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
Pre-reg	50	8.17	12.30	1.03	520.52	7.5	14.3	15.6	19.7
1	50	4.20	8.67	0.93	157.00	1.3	7.4	15.4	50.5
2	60	4.97	8.64	0.92	75.59	1.8	10.1	20.0	48.9
3	70	6.88	11.44	0.86	181.89	2.6	15.9	31.8	58.2
4	80	9.36	12.72	0.83	90.69	4.6	25.0	38.1	61.0
5	90	15.87	17.97	0.80	175.08	9.0	41.3	54.5	75.3
6	100	223.58	907.16	0.86	19863.81	57.1	327.9	800.2	3802.0
7	110	1249.07	2833.03	8.30	27911.80	321.8	1905.0	3914.0	15797.0

### 9.1.1 PX\_TRT-REG1: Time of the first register transaction(Registration use case) (Mean per second)



### 9.1.2 PX\_TRT-REG1: Time of the first register transaction(Registration use case) Histogram



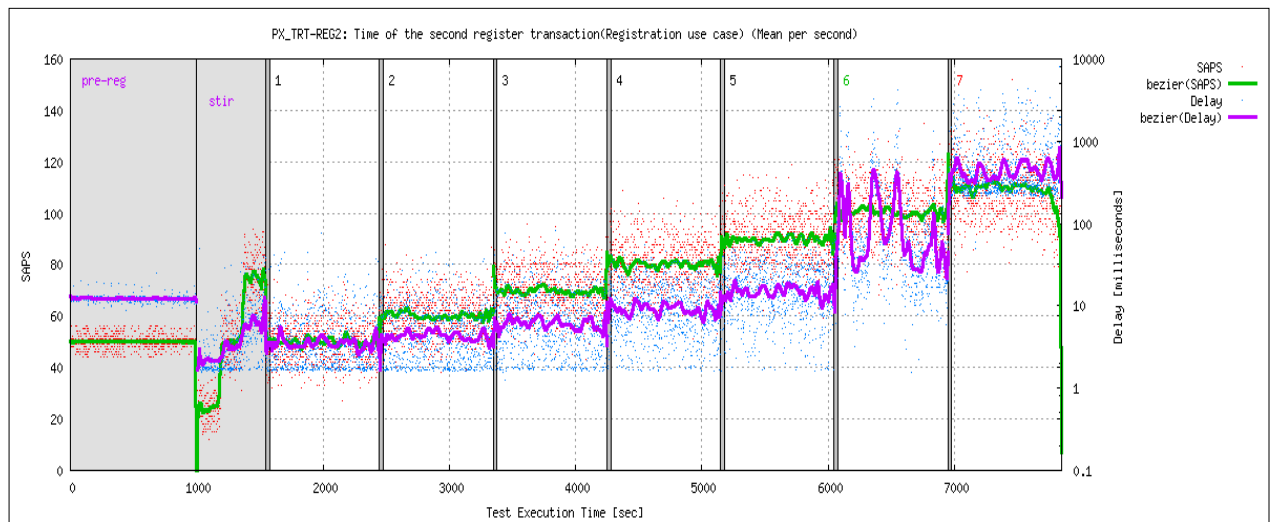
## 9.2 PX\_TRT-REG2: Time of the second register transaction

This graph represents the time of the second register transaction in the registration use\_cases, i.e. the delay between the second REGISTER and the 200 OK.

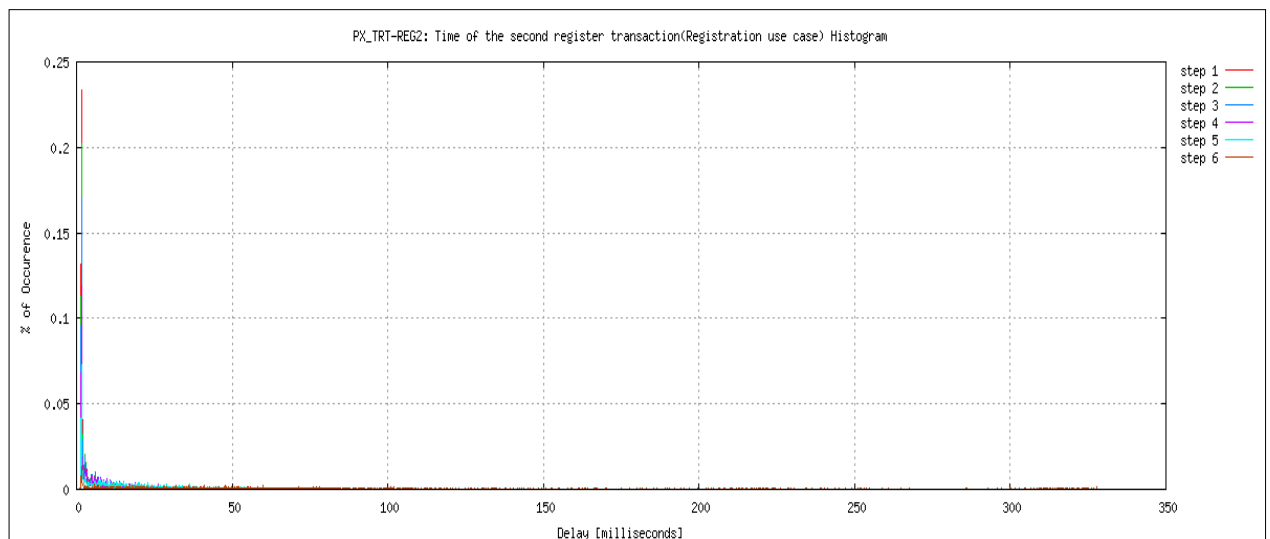


		PX_TRT-REG2 (msec)							
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
Pre-reg	50	12.33	4.54	1.46	508.05	12.6	15.8	17.1	20.7
1	50	4.72	7.06	1.54	78.85	1.8	9.9	16.2	34.9
2	60	5.85	7.91	1.11	84.17	2.5	13.7	21.8	36.7
3	70	8.39	11.29	1.22	136.94	3.3	21.0	32.8	50.2
4	80	11.70	13.29	1.18	109.64	6.5	29.5	39.0	58.9
5	90	19.64	19.05	1.29	120.08	13.5	47.1	58.5	81.6
6	100	161.69	586.61	1.24	15919.54	63.9	321.0	358.2	1791.0
7	110	614.79	1554.30	19.15	27742.13	289.6	812.8	1824.0	7855.0

### 9.2.1 PX\_TRT-REG2: Time of the second register transaction(Registration use case) (Mean per second)



### 9.2.2 PX\_TRT-REG2: Time of the second register transaction(Registration use case) Histogram

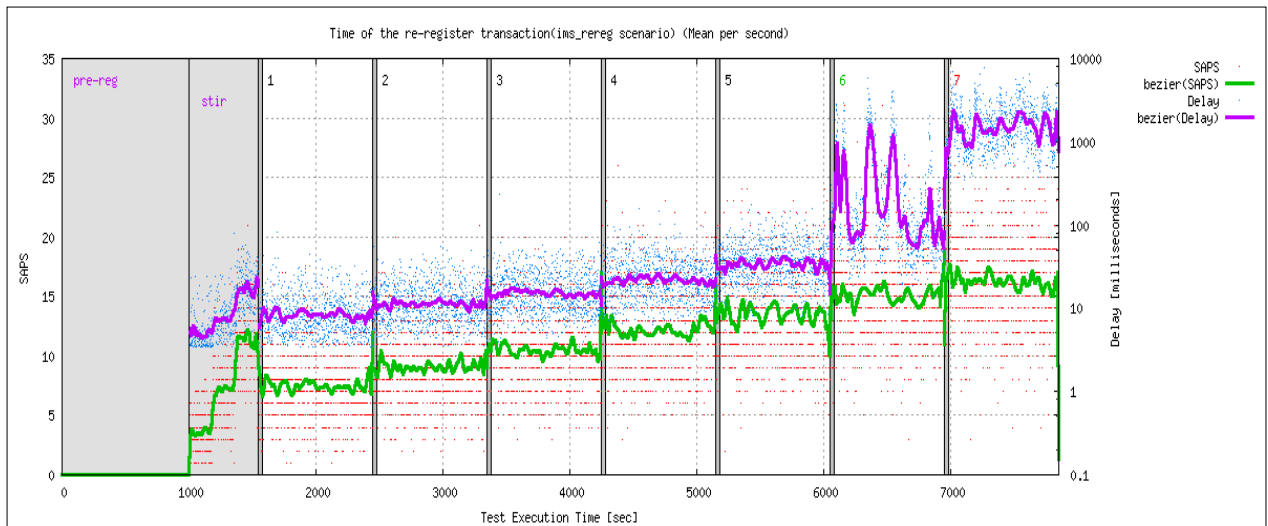


### 9.3 ims\_rereg : Time of the re-register transaction

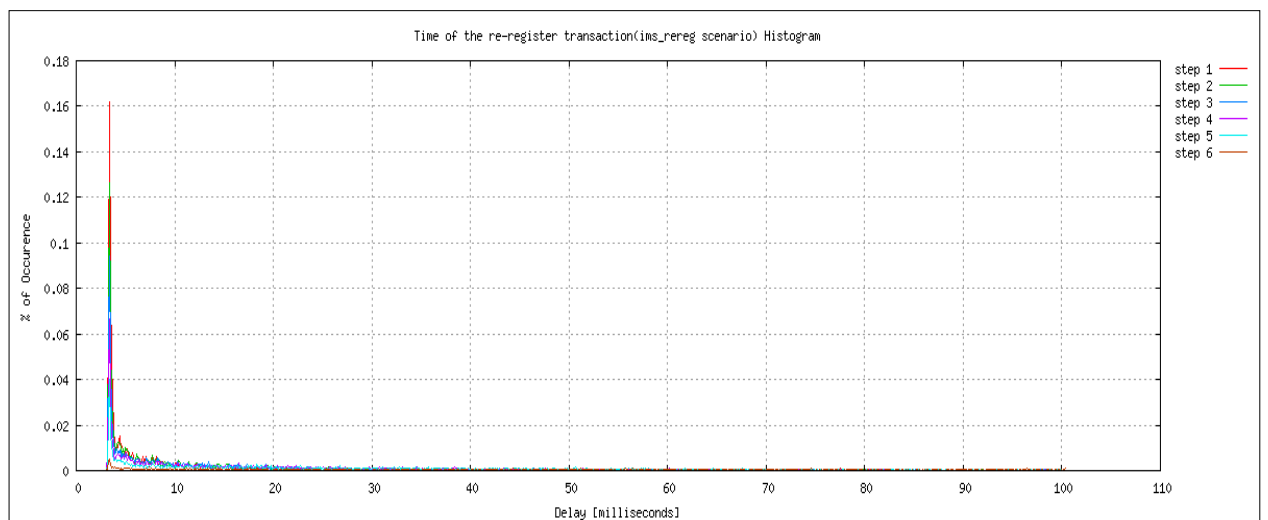
This graph represents the time of re-register transaction i.e. the time between the REGISTER and the 200 OK.

		PX_TRT-REG4 (msec)							
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
1	50	9.80	14.14	2.64	204.33	4.5	19.8	32.5	81.3
2	60	13.18	17.92	2.52	151.69	5.9	30.2	50.3	94.7
3	70	18.01	24.13	2.52	400.02	8.4	45.6	71.5	108.5
4	80	26.02	30.26	2.53	355.52	14.1	67.9	91.7	129.1
5	90	40.21	38.91	2.47	304.25	27.3	98.1	120.9	159.9
6	100	394.91	1183.40	2.50	35692.58	131.6	701.3	1192.0	4183.0
7	110	1731.55	2947.60	3.61	35647.00	687.5	4021.0	8035.0	16101.0

#### 9.3.1 Time of the re-register transaction(ims\_rereg scenario) (Mean per second)



#### 9.3.2 Time of the re-register transaction(ims\_rereg scenario) Histogram

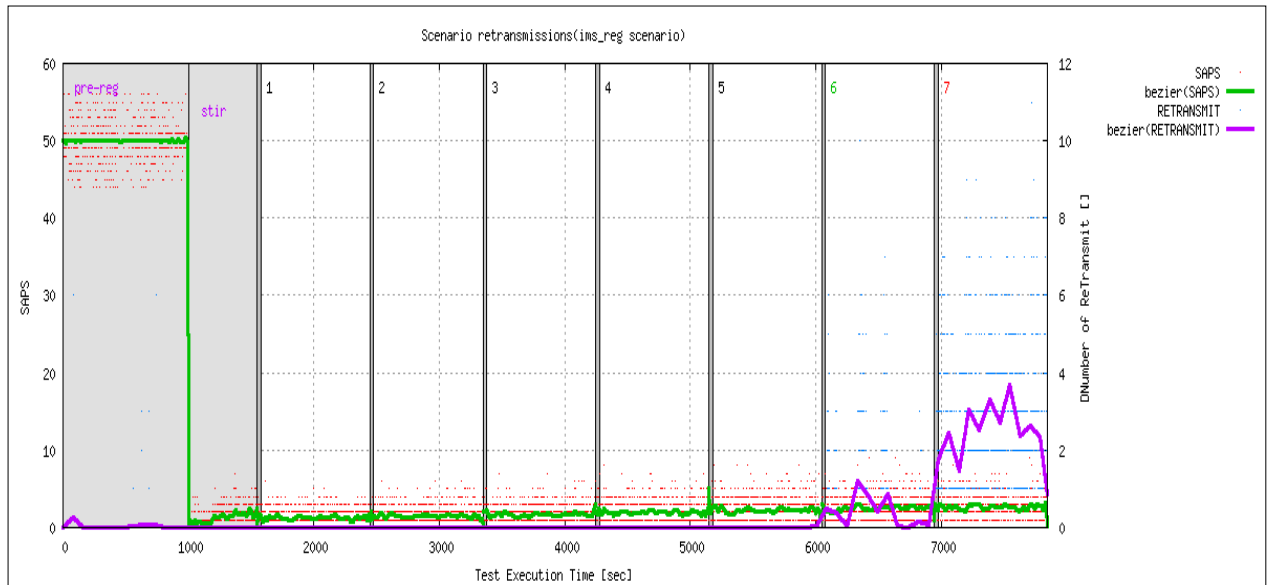


### 9.4 ims\_reg : Scenario retransmissions

This graph represents the number of retransmissions per seconds for this scenario.

Step	Requested Load	RETRANSMIT							
		Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
Pre-reg	50	0.03	0.35	0.00	6.00	0.0	0.0	0.0	0.0
1	50	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0
2	60	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0
3	70	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0
4	80	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0
5	90	0.00	0.00	0.00	0.00	0.0	0.0	0.0	0.0
6	100	0.41	1.10	0.00	10.00	0.0	2.0	3.0	6.0
7	110	2.70	2.01	0.00	11.00	2.0	6.0	6.0	8.0

#### 9.4.1 Scenario retransmissions(ims\_reg scenario) over time

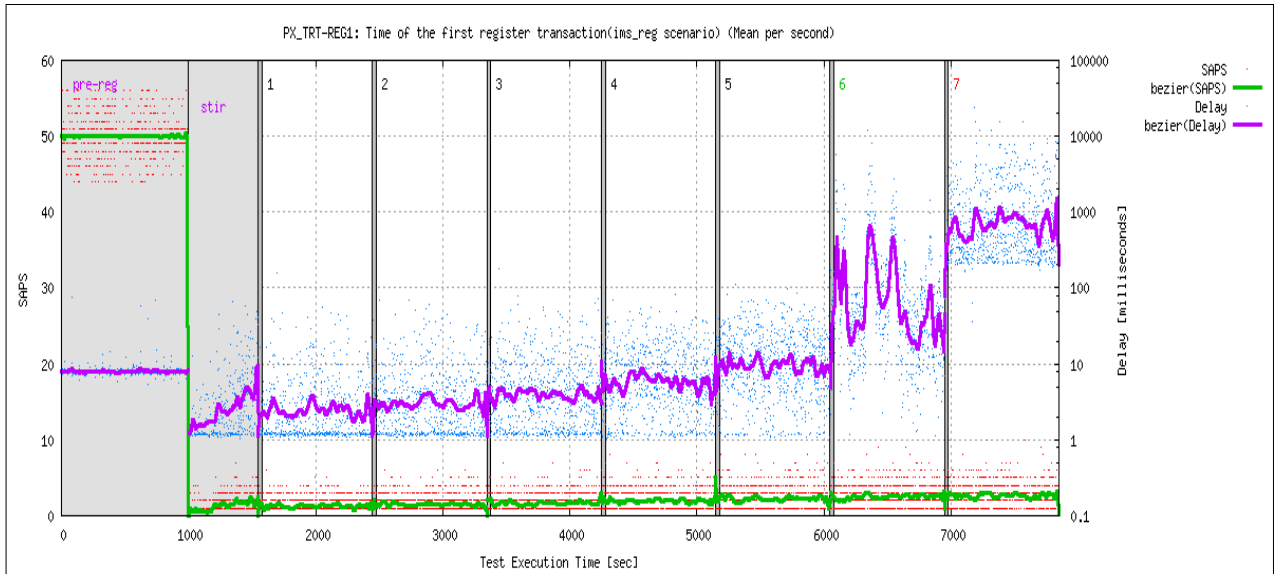


### 9.5 ims\_reg : PX\_TRT-REG1: Time of the first register transaction

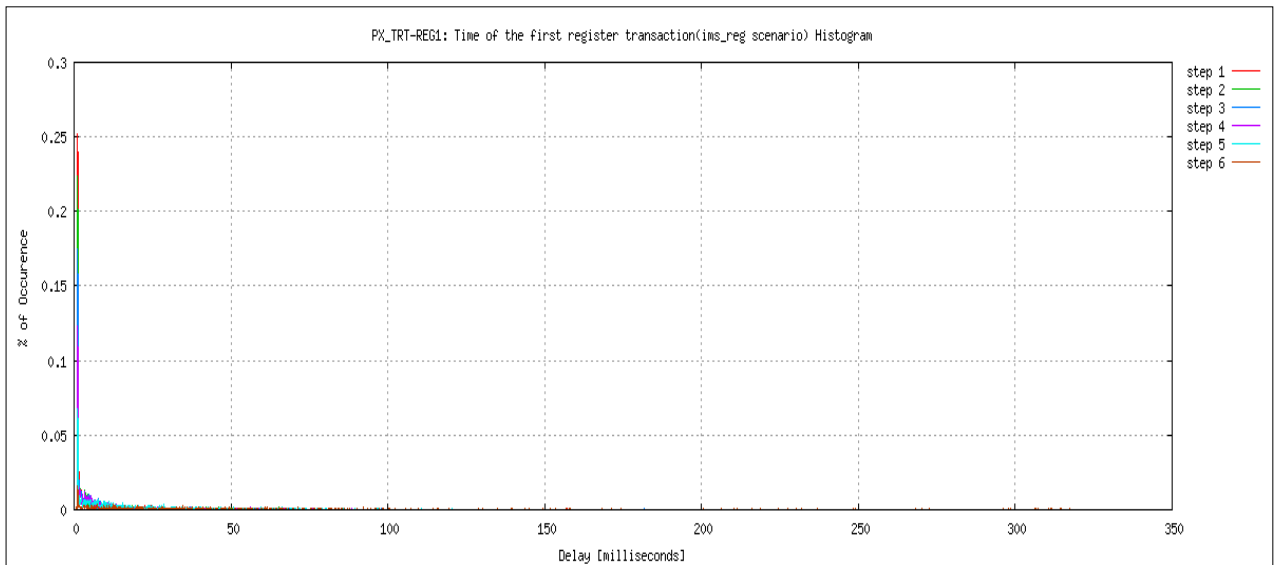
This graph represents the time of the first register transaction i.e. the time between the REGISTER and the 401 Unauthorized.

Step	Requested Load	PX_TRT-REG1 (msec)							
		Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
Pre-reg	50	8.17	12.30	1.03	520.52	7.5	14.3	15.6	19.7
1	50	4.33	9.12	0.96	157.00	1.2	7.7	18.7	47.0
2	60	5.21	9.53	0.99	75.59	1.6	10.7	22.7	51.6
3	70	6.94	12.05	0.93	181.89	2.5	15.3	33.6	59.7
4	80	9.19	12.46	0.83	88.57	4.5	23.2	37.1	59.7
5	90	15.34	17.51	0.80	120.48	8.8	40.1	54.2	75.7
6	100	233.80	928.21	0.88	15898.13	57.6	331.5	813.3	3836.0
7	110	1266.36	2791.03	8.30	27819.44	323.2	3720.0	7726.0	15797.0

### 9.5.1 PX\_TRT-REG1: Time of the first register transaction(ims\_reg scenario) (Mean per second)



### 9.5.2 PX\_TRT-REG1: Time of the first register transaction(ims\_reg scenario) Histogram

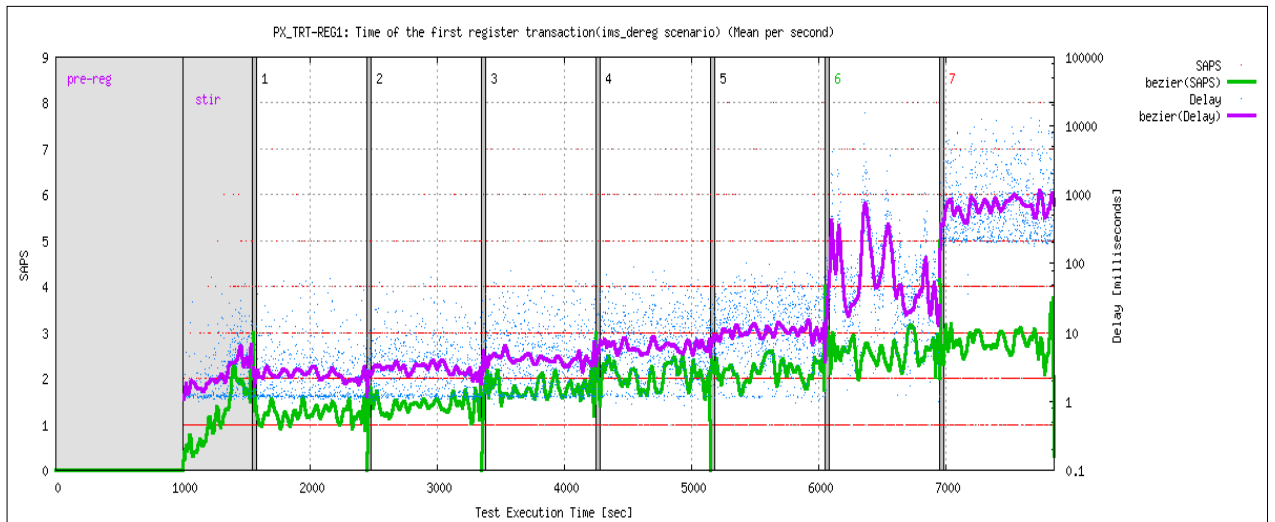


### 9.6 ims\_dereg : PX\_TRT-REG1: Time of the first register transaction

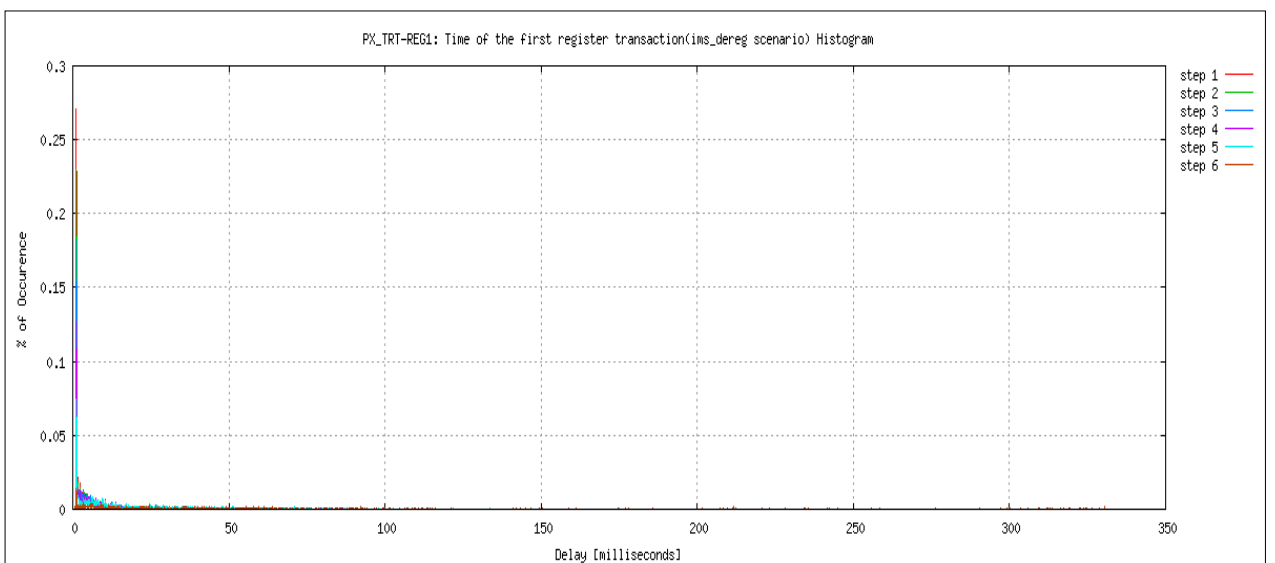
This graph represents the time of the first register transaction i.e. the time between the REGISTER and the 401 Unauthorized.

		PX_TRT-REG1 (msec)							
Step	Requested Load	Mean	Standard Deviation	Minimum	Maximum	Percentile 50	Percentile 90	Percentile 95	Percentile 99
1	50	4.06	8.18	0.93	64.27	1.3	6.7	12.3	53.3
2	60	4.72	7.59	0.92	71.30	1.9	9.6	17.5	43.6
3	70	6.82	10.84	0.86	104.52	2.6	16.4	30.2	55.0
4	80	9.53	12.97	0.89	90.69	4.7	25.8	39.3	61.7
5	90	16.41	18.41	0.85	175.08	9.4	43.4	54.8	75.0
6	100	213.69	886.22	0.86	19863.81	56.9	326.0	734.4	3734.0
7	110	1233.34	2872.25	9.09	27911.80	320.8	1882.0	3881.0	15793.0

**9.6.1 PX\_TRT-REG1: Time of the first register transaction(ims\_dereg scenario) (Mean per second)**



**9.6.2 PX\_TRT-REG1: Time of the first register transaction(ims\_dereg scenario) Histogram**



## Appendix

The test was run based on the following IMS benchmark parameters.

Parameter Name	Parameter Value	Parameter Info
StepTransientTime	30	Time after the start of a step for which data is ignored (in seconds)
PreRegistrationMaxIHS	1	Accepted percentage of inadequately handled registrations during the pre-registration preamble phase (in percent)
StirTime	5	Duration of stir phase where the SUT is gradually brought up to a load of 'InitialSAPS' (in minutes)
StirMaxIHS	1	Accepted percentage of inadequately handled scenarios during the stir phase (in percent)
StepTime	30	Duration of each step of the run (in minutes)
InitialSAPS	100	Load applied to the SUT at the first step (in SAPS)
StepNumber	3	Number of steps in the actual benchmark phase of the run
StirSteps	3	Number of steps in the stir phase
PreRegistrationRate	100	Registration rate during the pre-registration preamble phase (in registration attempts per seconds)
SAPSIIncreaseAmount	10	System load increase at each step (in SAPS)
TransportTCP	0	Use TCP-mode IMS Benchmark SIPp
MaxTimeOffset	0	Maximum time offset (in microseconds) allowed between any Test System and the manager. This is checked at the beginning of the benchmark in order to avoid doing a run with invalid time measurements. 0 means no check (not recommended).
ExecuteSIPp	0	Whether or not (0/1) the SIPp instances should be started by the manager. If enabled, the SIPp instances run in the background and their output cannot be viewed. If disabled, the SIPp instances must be started manually.
ManagerIP	10.20.106.106	IP Address of the Manager
RingTimeDistr	exponential	Random distribution of the ringing time (poisson/exponential)
RegistrationExpire	1000000	Registration timeout that emulated users will request (in seconds)
PMMDDataSize	140	Average length of page-mode messages (in bytes)
HoldTimeDistr	exponential	Random distribution of the call hold time (poisson/exponential)
PMMDDataSizeDistr	uniform	Random distribution of the length of page-mode messages (uniform)
HoldTime	30	Average call hold time (in seconds)
RingTime	5	Average ringing time (in seconds)
PublicIdentityFormat	subs%06d	Format used to generate public identity of users. The %d parameter gets replaced by the index of the subscriber.
PrivateIdentityFormat	subs%06d	Format used to generate private identity of users. The %d parameter gets replaced by the index of the subscriber.
DontPreRegisterButUseSippIP	0	Do not include a pre-registration phase. Use the SIPp Test System IP Address as user domain instead.
TotalProvisionedSubscribers	100000	Total number of provisioned subscribers
UserRealm	open-ims.test	Primary domain to which provisioned users belong (for authentication purpose)
UserDomain	open-ims.test	Primary domain to which provisioned users belong
PercentRoamingSubscribers	0	
PercentRegisteredSubscribers	80	Percentage of subscribers who will be pre-registered during the preamble phase (in percent)
UserPasswordFormat	abcdefgh	Format used to generate passwords identity of users. If present, a %d parameter gets replaced by the index of the subscriber.

The following information is also available for the test

Parameter Name	Parameter Value	Parameter Info
rand_seed	1271889544	Value used to initialize the random number generators
prep_offset	2000	Time (ms) for scenario preparation (user reservation, etc.) prior to actual execution
highest_measured_time_offset	345	Highest time offset observed at startup between any test system and the manager (microseconds)

System	Command Line
TS1	./sipp -id 1 -i 10.20.106.107 -user_inf ./ims_users_1.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS2	./sipp -id 2 -i 10.20.106.20 -user_inf ./ims_users_2.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS3	./sipp -id 3 -i 10.20.106.21 -user_inf ./ims_users_3.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS4	./sipp -id 4 -i 10.20.106.22 -user_inf ./ims_users_4.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS5	./sipp -id 5 -i 10.20.106.23 -user_inf ./ims_users_5.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS6	./sipp -id 6 -i 10.20.106.24 -user_inf ./ims_users_6.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS7	./sipp -id 7 -i 10.20.106.25 -user_inf ./ims_users_7.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS8	./sipp -id 8 -i 10.20.106.26 -user_inf ./ims_users_8.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS9	./sipp -id 9 -i 10.20.106.27 -user_inf ./ims_users_9.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
TS10	./sipp -id 10 -i 10.20.106.28 -user_inf ./ims_users_10.inf -rmctrl 10.20.106.106:5000 10.20.106.105:5060 -trace_err -trace_cpumem -trace_scen -trace_retrans
Manager	./manager -f ims_bench_1/manager.xml
SUT 1	./cpum 10.20.106.106:5000

## APPENDIX F

KEMARI (<http://www.xen.org/community/projects.html>)

- **Introduction**

- **Kemari :**

- **Aim : Keeps VMs transparently running at times of HW failures**
- No special HW required
- No modification to applications
- Takes advantage of both **checkpointing** and **Lockstep**

### 1. Design and Implementation

- **Synchronization Model**

- **Kemari Procedure**

- Synchronizes state of VMs on multiple HW (Primary, Secondary VMs kept in sync)
- Detects HW failures
- **Transfers the state of “Primary VM” to “Secondary VM”, when an event occurs**
- Synchronizes VMs when the “Primary VM” is about to send an event to devices such as storage and networks
- Switches to a Secondary VM, when failure occurs on a Primary VM
- Does not require external buffering mechanisms which impose output latencies ( which is necessary in continuous checkpointing )

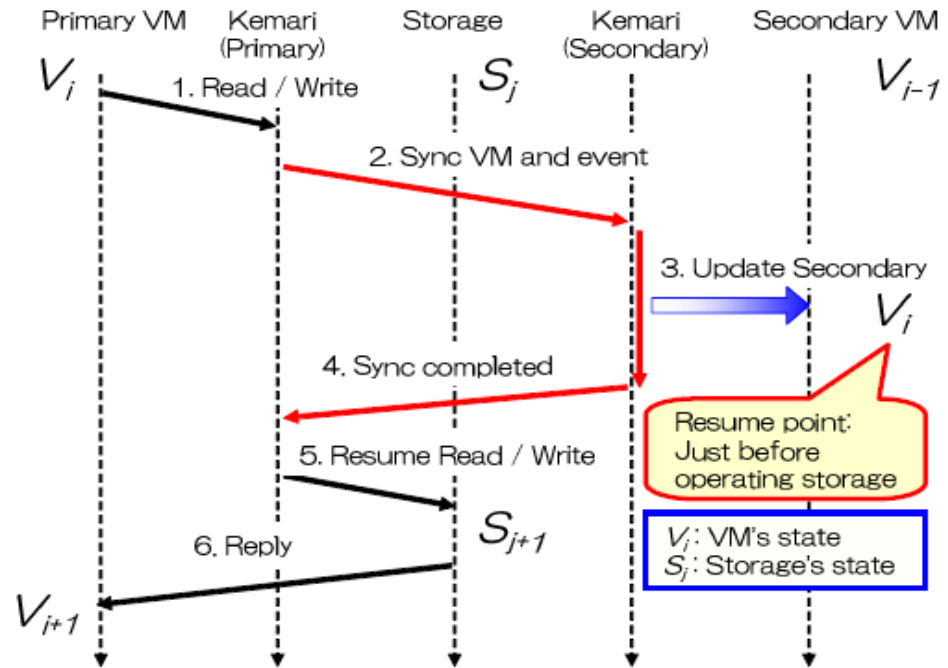


Figure: Synchronization steps initiated by an event sent to storage

- > **Scenario:** Primary VM tries to read/write to the storage,
- > This action outputs an event to the storage
- > Since this event will change the state of the storage, Kemari
  - Pauses the Primary VM
  - Updates the Secondary VM state to the current state of the Primary VM
  - Resumes the primary VM ( after synchronization )
  - Transmits the trapped event to the storage
- *Secondary VM can continue transparently, when the Primary VM FAILS.*
- > If a HW failure is detected, Kemari
  - switches to the Secondary VM
  - starts running from the latest **synchronized point** ( where **Secondary VM** and **virtual disk** are consistent )
- *Network Events are handled in a similar manner*
- **Applications using reliable protocols (TCP)**
  - transparently continue after switching to the Secondary VM.
- **Applications using unreliable protocols (UDP) :**
  - need to keep the network status consistent by themselves.
- **Implementation**



Kemari is implemented based on Xen VMM.

Two types of domains:

- **Privileged Domain (Domain 0)**: can access the devices.
  - **Unprivileged Guest Domain** : need to request I/O through virtualized devices such as **virtual disks** and **virtual networks**, provided by **domain 0** .
  - *Frontend* : **guest domain** side of the **virtual device**
  - *Backend* : **domain 0** side of the **virtual device**
  - *Event channel* : *notification mechanism* connecting the *frontend* and the *backend*, provided by Xen.
- > Kemari is implemented across VMM and *userland*.
  - > The VMM part of Primary Kemari captures events sent through the “event channel” to kick start the “synchronization” process.
  - > When Kemari detects an event from the *frontend*,
    - > *Pauses* the *guest domain*
    - > Searches for *dirty pages* in the *guest domain* created since the previous synchronization
    - > Notifies *userland* part of Kemari to send the pages and the *vcpu* context to the Secondary Kemari.
    - > *Unpauses* the *guest domain* and sends the captured event to the *backend* (after the synchronization).
  - Kemari used a guest domain whose page tables are virtualized by *shadow pagetables*. With this technique, Kemari avoided depending on *suspend* used in the live migration, which might change the state of the guest.
  - It also simplified the transfer mechanism by removing the canonicalization process of the *pagetables*.

## APPENDIX G

REMUS (<http://people.cs.ubc.ca/~andy/papers/remus-nsdi-final.pdf> )

### 1. Approach in brief

- Migration of running VMs between physical hosts (Virtualization aids to create a copy of a running VM)
- Encapsulates **Protected SW in a VM**
- Replicate snapshots of an entire running OS instance between a pair of physical machines
- Uses **speculative execution** to run “Active VM” ahead of **replicated system state**
- **Single host to execute speculatively** (rather than running two hosts in *lock-step*) then
  - *checkpoint*
  - *replicate* its state asynchronously
- **System State** : not made externally visible until the *checkpoint* is committed
- **External output** : Not released until the system state is replicated ( Network packets etc.)

### 2. Goals

- **Generality** : HA provided as a low-level service and independent of the protected Application or HW
- **Transparency** : *OS or Applications not tied to HA logic*
- **Seamless Failure Recovery**
  1. Host State - **Preserves** completely (such as active network connections )
  2. Downtime - few seconds
  3. Failure recovery proceeds rapidly
  4. Appears as a temporary packet loss from the external user's perspective

### 3. Design and Implementation

Based on “Xen VMM” and extends Xen’s support for “live migration” to provide “*fine-grained checkpoints*”.

- Propagates *checkpoints of Active VM to Backup physical host* frequently.
- On the Backup, VM image is in memory and may begin execution immediately if Active system FAILS.
- Since Backup is only periodically consistent with the Primary, all network output must be buffered until state is synchronized on the Backup.

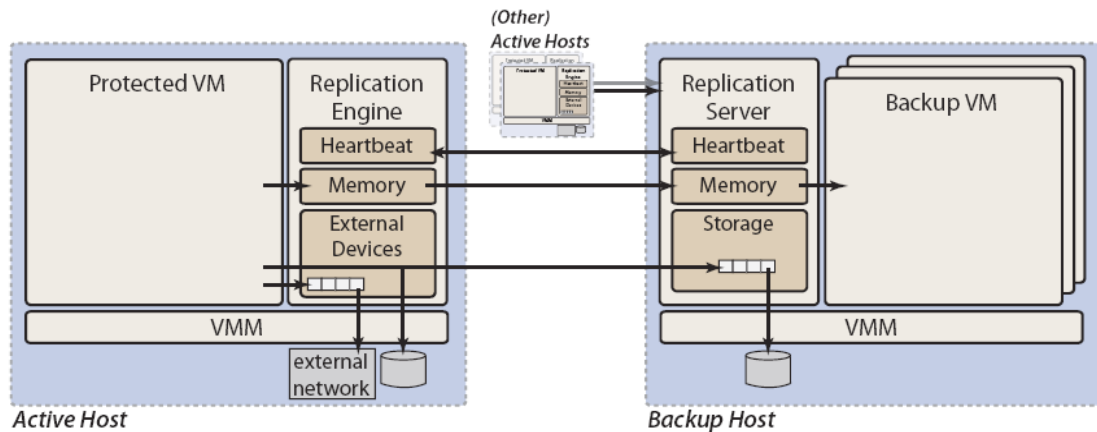


Figure 2: Remus: High-Level Architecture

- To maintain disk replication,
  - Writes to the Primary disk are transmitted asynchronously to the Backup
  - Buffered in Backup's RAM until the corresponding memory checkpoint arrives
  - At that point,
    - complete *checkpoint* is acknowledged to the Primary
    - Primary releases outbound Network traffic
    - buffered disk writes are applied to the Backup disk
- The checkpoint, buffer, and release cycle happens very frequently
  - whole-machine checkpoint including network - at frequencies up to 40 times/sec
  - on disk state - every 25 milliseconds.

## 1. Failure model

Remus provides the following properties:

- The fail-stop failure of any single host is tolerable.
- **Protected system's data will be left in a crash-consistent state if both Primary and Backup Hosts FAIL concurrently.**
- No output will be made externally visible until the associated system state has been committed to the replica.
- **Existing commercial HA products :**
  - If a physical host fails, reboots the VM on another host from its crash-consistent disk state
- **Remus:**
  - Survives **failure** on time frames similar to those of live migration
  - Keeps VM running and Network connections intact
  - Exposed state is not lost and disks are not corrupted
  - Host Machines connected over redundant Gigabit Ethernet connections and survives the failure of any one of these components.
  - By incorporating block devices into its state replication protocol, it avoids requiring expensive, shared network-attached storage for disk images.

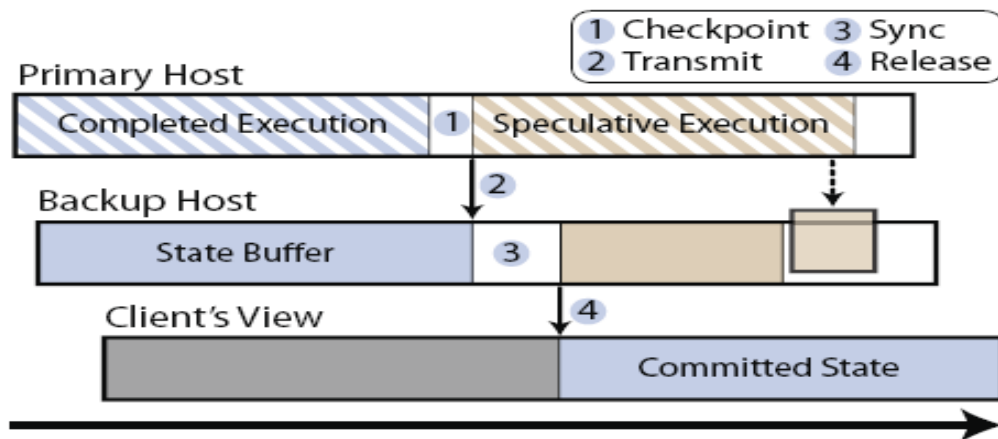


Figure 1: Speculative execution and asynchronous replication in Remus.

- **Pipelined Checkpoints**

- Once per epoch,
  - Pause the running VM
  - copy any changed state into a buffer (stop and copy stage of live migration)
  - With state changes preserved in a buffer, VM is unpaused and speculative execution resumes.
- Buffered state is transmitted and stored in memory on the Backup host.
- checkpoint is acknowledged to the Primary
- Finally, buffered network output is released.

## 2. Memory and CPU

- Checkpointing is implemented above Xen's existing machinery for performing "live migration".
- Remus implements checkpointing as repeated executions of the final stage of "live migration"

- **Modifications to the migration process described below**

- **Migration enhancements**

In live migration,

- guest memory is iteratively copied over a number of rounds – may consume minutes of execution time
- capturing frequent VM checkpoints - *every* checkpoint is just the final stop-and-copy phase of migration
- **Remus optimizes checkpoint signaling in following two ways:**
  - *reduces number of inter-process requests required to suspend/resume guest domain*
  - *entirely removes xenstore from the suspend/resume process*

- Migration process desired to **suspend** a VM
- XEN code ( original )

- sent a message to *xend* ( *VMmanagement* daemon )
  - *xend* in turn wrote a message to *xenstore*
  - *xenstore* alerted the *guest* by an event channel that it should *suspend* execution
  - **Guest’s final act before suspending was**
    - to make a hypercall which descheduled the domain and caused **Xen** to send a notification to *xenstore*
    - *xenstore* then sent an interrupt to *xend*
    - *xend* finally returned control to the **migration process**
    - This convoluted process takes an arbitrary amount of time - latencies in the range of 30 to 40ms
- **Remus’s optimized suspend code (of Xen)** streamlines this process as follows
  - **signaling changes**
    - **event channel in guest** - to receive **suspend** requests - migration process can invoke directly.
    - **new hypercall** - to allow processes to “register an event channel for callbacks notifying them of the completion of VM suspension”.
    - above “notification mechanisms” reduce time required to **suspend** a VM to about 100 microseconds
  - **memory copying process**
    - quickly filters out clean pages from the memory scan - most memory is unchanged between rounds.
    - maps the guest domain’s entire physical memory into the replication process when it begins, rather than mapping and unmapping dirty pages at every epoch
  - **Checkpoint support**
    - **checkpoint support in Xen** - two changes to existing **suspend-to-disk** and **live migration code** :
      - » support for resuming execution of a domain after it had been suspended  
( *Xen previously did not allow “live checkpoints” - destroyed the VM after writing its state out* )
      - » **suspend** program was converted from a one-shot procedure into a daemon process.
    - **Resume support** :
      - » A new hypercall to mark the domain as schedulable again
      - » A similar operation is necessary in order to rearm watches in *xenstore*
  - **Asynchronous transmission.**

To allow the guest to resume operation quickly - migration process was modified to copy touched pages to a staging buffer rather than delivering them directly to the network ( while domain is paused )
  - **Guest modifications**

Paravirtual guests in Xen contain a suspend handler that cleans up device state upon receipt of a suspend request. The suspend request handler has also been modified to reduce the amount of work done prior to suspension. In the original code, suspension entailed disconnecting all devices and

unplugging all but one CPU. This work was deferred until the domain was restored on the other host.

### 3. Network buffering

- **Inbound** traffic
  1. delivered to protected host immediately
- **Outbound** packets generated since the previous checkpoint are queued until
  1. current state has been checkpointed
  2. checkpoint has been acknowledged by the Backup site

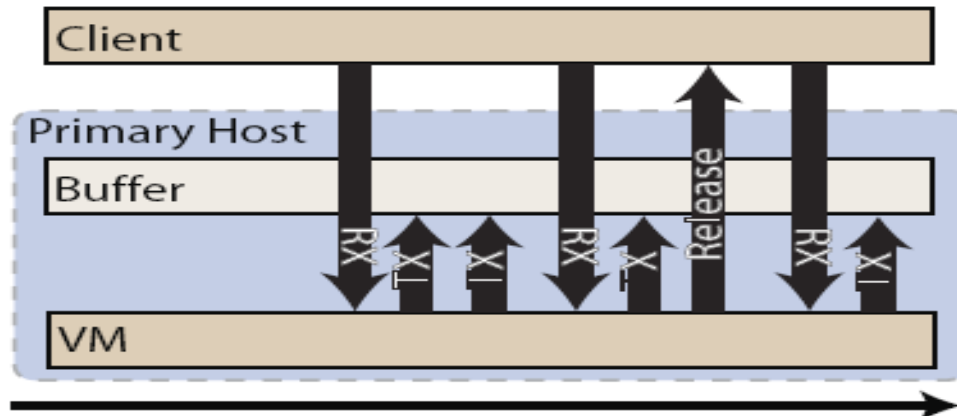


Figure 3: Network buffering in Remus.

Remus implemented this buffer as a linux queuing discipline applied to the guest domain's network device in *domain 0*, which responds to two RT netlink messages.

1. Before the guest is allowed to resume execution after a checkpoint, the network buffer receives a CHECKPOINT message, which causes it to insert a barrier into the outbound queue preventing any subsequent packets from being released until a corresponding release message is received.
2. When a guest checkpoint has been acknowledged by the backup, the buffer receives a RELEASE message, at which point it begins dequeuing traffic up to the barrier.

Two minor wrinkles in this implementation.

1. Linux queuing disciplines only operate on *outgoing* traffic. Under Xen, guest network interfaces consist of a frontend device in the guest, and a corresponding backend device in *domain 0*. Outbound traffic from the guest appears as *inbound* traffic on the backend device in domain 0. Therefore in order to queue the traffic, we convert the inbound traffic to outbound by routing it through a special device called an *intermediate queuing device*. This module is designed to work at the IP layer via *iptables*, but it was not difficult to extend it to work at the bridging layer.
2. The second wrinkle is due to the implementation of the Xen virtual network device. For performance, the memory used by outbound networking traffic is not copied between guest domains and domain 0, but shared.

- However, only a small number of pages may be shared at any one time. If messages are in transit between a guest and domain 0 for only a brief time, this limitation is not noticeable.
- Unfortunately, the network output buffer can result in messages being in flight for a significant amount of time, which results in the guest network device blocking after a very small amount of traffic has been sent.
- Therefore when queueing messages, Remus “first copies them into local memory and then release the local mappings to shared data”.

#### 4. Disk buffering

- Remus recovers from a single host failure.
- Preserves crash consistency to recover even if *both* hosts fail.
- Maintains a complete mirror of the Active VM’s disks on the Backup host.
- Before Engaging Protection : **Current State of disk on Primary is mirrored to the Backup host**
- After Engaging Protection : **Writes to persistent storage are tracked and checkpointed** (similar to updates to memory)

Figure 4 : gives a high-level overview of the disk replication mechanism

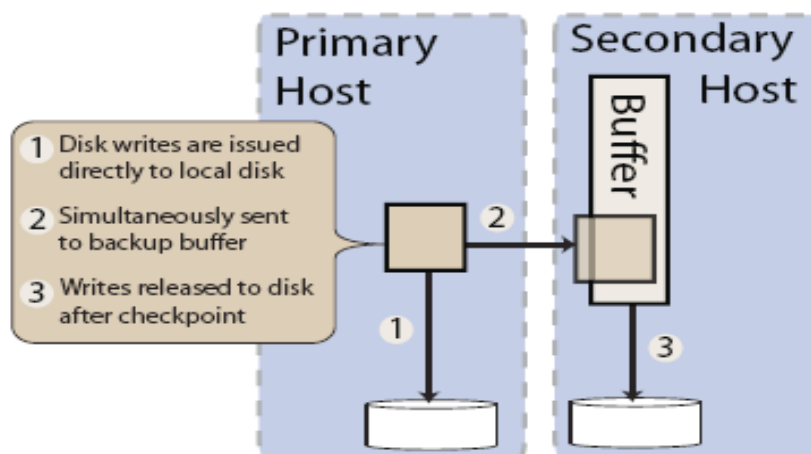


Figure 4: Disk write buffering in Remus.

- Writes to disk from the Primary are treated as write-through: they are immediately applied to the Primary disk image, and asynchronously mirrored to an in-memory buffer on the Backup.
- This approach provides two direct benefits:
  - “**Active disk image remains crash consistent**” at all times
  - writing directly to disk accurately accounts for the latency and throughput characteristics of the physical device.
- Disk updates reside in memory until the Backup acknowledges a checkpoint
- Disk state unchanged until the entire checkpoint has been received – so that Backup can **rollback** to the most recent complete checkpoint.
- Disk request buffer may be applied to Disk, once the checkpoint is acknowledged

- In the event of a failure, Remus will wait until all buffered writes have been applied before resuming execution.
- Only one of the two disk mirrors managed by Remus is actually valid at any given time. This point is critical in recovering from multi-host crashes. This property is achieved by the use of an activation record on the backup disk, which is written after the most recent disk buffer has been completely flushed to disk and *before* the backup VM begins execution. In recovering from multiple host failures, this record may be used to identify the valid, crash consistent version of the disk.
- **Disk Buffer** : implemented as a Xen block tap module.
- **Block tap** : device which allows a process in the **privileged domain** to efficiently interpose itself between the frontend disk device presented to a guest VM and the backend device which actually services requests.
- The buffer module logs disk write requests from the protected VM and mirrors them to a corresponding module on the backup, which executes the checkpoint protocol described above and then removes itself from the disk request path before the Backup begins execution in the case of failure at the Primary.

## 5. Detecting Failure

A simple failure detector is directly integrated in the checkpointing stream:

- **Timeout**
  - Backup NOT responding to commit requests - Primary assumes that the “Backup has crashed” and disables protection.
  - Primary not transmitting **new checkpoints** - Backup assumes that the Primary has crashed and resumes execution from the most recent checkpoint.
- The system is configured to use
  - a pair of bonded network interfaces
  - two physical hosts are connected using a pair of Ethernet crossover cables (or independent switches) on the protection NICs.
  - Should both of these network paths fail, Remus does not currently provide mechanism to fence execution.