**Performance Measurements and Modeling
Of a Java-based
Session Initiation Protocol Implementation**

by

Danny Hok-Ching Lau

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of

Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

Ottawa, Ontario, Canada, K1S 5B6

August 28, 2006

© Copyright 2006, Danny Hok-Ching Lau

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-18322-9

Our file Notre référence

ISBN: 978-0-494-18322-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Session Initiation Protocol (SIP) is a new technology that is becoming more popular as consumers move from the traditional Public Switched Telephone Network (PSTN) to Voice over IP (VoIP). One of the main concerns with this technology is the system's performance and how it compares to the PSTN system. In this thesis, performance analysis was performed on an IBM's SIP implementation prototype in a standalone and cluster environment with the use of network tracing and performance modeling on each environment. This process includes gathering data from test runs and extracting performance parameters from Ethereal packet traces to build performance models. After constructing the models based on a template, calibration is required in order to match the model prediction with real system test data. Using the calibrated models, some bottlenecks were identified and suggestions to improve the overall maximum throughput were developed and were subsequently implemented into the system.

Acknowledgements

First of all, I would like to thank my supervisor Professor Greg Franks for all the guidance during the past three years with my thesis work. It has been an honor working with you for the past few years. I would also like to thank Professor Murray Woodside and Tauseef Israr for letting me to be part of their research during this time.

I want to thank Curtis Hrischuk and my manager Jay Tunkel for giving me the opportunity to work with them and give me guidance during my internship at IBM during the year of 2005. I would also like to thank my co-worker Mohit Jain for helping me and working with me during that whole year. My experience at IBM has been extremely beneficial to me.

I would like to thank my family for their constant support during this time. Also, I would like to thank David and Joyce Nguyen and Leon Ip for their encouragements along the way. I would also like to thank Stephanie Wong for giving me insights on some of the topics and for proofreading my paper. Finally, I would like to thank everybody who has been with me every step of the way.

TABLE OF CONTENTS

CHAPTER 1.0: INTRODUCTION.....	1
1.1 NEED FOR PERFORMANCE ON SIP	2
1.2 RELATED WORK	3
1.3 CONTRIBUTION	5
1.4 OUTLINE OF THE THESIS	6
 CHAPTER 2.0: BACKGROUND	8
2.1 SIP PROTOCOL AND ARCHITECTURE.....	8
2.1.1 <i>SIP Architecture</i>	9
2.1.2 <i>SIP Protocol Overview</i>	12
2.1.3 <i>SIP Messages</i>	14
2.1.4 <i>SIP Header</i>	17
2.2 SOFTWARE PERFORMANCE ENGINEERING.....	18
2.2.1 <i>Performance Modeling</i>	20
2.2.2 <i>Ethernet Modeling and Simulation</i>	23
2.3 SIP SERVER TEST COMPONENTS.....	26
2.3.1 <i>SIPP Client</i>	27
2.3.2 <i>IBM WebSphere Application Server (WAS)</i>	28
2.3.3 <i>Back-to-Back User Agent</i>	30
2.4 PERFORMANCE TOOLS	31
2.4.1 <i>Performance Profiling Tools</i>	32

2.4.2	<i>System Monitoring</i>	34
2.4.3	<i>LAN Analyzer</i>	36
CHAPTER 3.0: SIP PERFORMANCE BENCHMARKING		38
3.1	SIP PERFORMANCE BENCHMARK: SIPSTONE.....	38
3.2	TEST ENVIRONMENT SETUP	41
3.2.1	<i>Hardware Configuration and Setup</i>	42
3.2.2	<i>Load Driver and SIP Server Software</i>	46
3.3	SIP INVITE SCENARIOS	48
3.3.1	<i>Standalone Invite Scenario</i>	48
3.3.2	<i>Cluster Invite Scenario</i>	52
3.3.3	<i>Invite Scenario Exceptions</i>	54
3.4	PERFORMANCE ENGINEERING TOOLKIT (PET)	55
3.5	PERFORMANCE TEST RUNS	59
CHAPTER 4.0: ANALYTIC MODEL CONSTRUCTION		63
4.1	BUILDING THE PERFORMANCE MODELS	63
4.1.1	<i>Standalone Environment Model Construction</i>	65
4.1.2	<i>Cluster Environment Model Construction</i>	69
4.2	MODEL PARAMETERS	74
4.2.1	<i>Ethereal Packets</i>	74
4.2.2	<i>Algorithm for Extracting Service Times</i>	76
4.2.3	<i>Obtaining Performance Numbers</i>	80

4.2.4	<i>Calculating Network Load</i>	85
4.3	SOLVING THE PERFORMANCE MODELS	90
4.4	MEASURED PERFORMANCE DATA	93
4.5	MODEL ANALYSIS	95
 CHAPTER 5.0: MODEL CALIBRATION AND SYSTEM PREDICTIONS		99
5.1	FACTORS THAT MIGHT AFFECT RESULTS	99
5.2	OBJECTIVE OF MODEL CALIBRATION.....	101
5.3	CALIBRATING THE MODELS	102
5.3.1	<i>Standalone Environment Calibration</i>	103
5.3.2	<i>Cluster Environment Calibration</i>	105
5.4	CALIBRATED MODEL ANALYSIS	107
5.5	MAXIMUM GIGABIT NETWORK CAPACITY.....	109
5.6	CAPACITY PLANNING AND SYSTEM IMPROVEMENTS	115
 CHAPTER 6.0: CONCLUSIONS		123
6.1	CONTRIBUTIONS	124
6.2	FUTURE WORK	125
 REFERENCES		127
 APPENDIX A-1: ETHEREAL TRACE PACKET WITH SIP HEADERS		132
 APPENDIX A-2: ETHEREAL TRACE SUMMARY OF A COMPLETE CALL FLOW		131

APPENDIX A-3: ETHERREAL TRACE OF RETRANSMISSION/FAILURE	132
APPENDIX B-1: ORIGINAL STANDALONE LQN MODEL	133
APPENDIX B-2: ORIGINAL CLUSTER LQN MODEL	134
APPENDIX B-3: CALIBRATED STANDALONE LQN MODEL	136
APPENDIX B-4: CALIBRATED CLUSTER LQN MODEL	137
APPENDIX B-5: OPEN CALIBRATED CLUSTER LQN MODEL	139
APPENDIX B-6: LQN MODEL FOR FULL CONFIGURATION.....	141
APPENDIX C-1: ETHERREAL TRACE PARSING SCRIPT – ANALYZE.SH ...	143
APPENDIX C-2: ETHERREAL TRACE PARSING SCRIPT – PACKET.SH	145
APPENDIX C-3: ETHERREAL TRACE PARSING SCRIPT – PACKETSLSP.SH	148
APPENDIX D-1: ORIGINAL STANDALONE LQN MODEL OUTPUT	152
APPENDIX D-2: ORIGINAL CLUSTER LQN MODEL OUTPUT	156

LIST OF FIGURES

Figure 1: Simple Internet Multimedia Protocol Stack	10
Figure 2: Simple SIP Session Example	12
Figure 3: User Agent Client Scenario	40
Figure 4: User Agent Server Scenario	41
Figure 5: Test Environment Setup	43
Figure 6: Deployment Diagram for Standalone Environment.....	45
Figure 7: Deployment Diagram for Cluster Environment.....	46
Figure 8: SIP Invite Scenario.....	49
Figure 9: SIP Standalone Invite Call Flow	51
Figure 10: SIP Cluster Invite Call Flow	53
Figure 11: Sample PET Graph Showing CPU Utilization.....	57
Figure 12: CPU Server Running at Ideal Condition.....	61
Figure 13: Sequence Diagram of First Interaction.....	66
Figure 14: LQN Model of the First Interaction	66
Figure 15: SIP Standalone Model.....	68
Figure 16: SIP Cluster Model.....	72
Figure 17: Performance Numbers Extraction Process.....	75
Figure 18: First set of Cluster Interaction.....	78
Figure 19: LQN Solver Output from the Original Standalone Model.....	91
Figure 20: LQN Solver Output from the Original Cluster Model.....	92
Figure 21: Calibrated Standalone Model Output.....	104

Figure 22: Calibrated Cluster Model Output	106
Figure 23: Open Model of the Calibrated Cluster Model at Maximum Throughput.....	113
Figure 24: CPU Utilization for Cluster with 1 Load Balancer and 3 SIP Servers.....	117
Figure 25: CPU Utilization of the Fully Utilized SIP Cluster System	119
Figure 26: LQN Model of Fully Configured SIP Model	121

LIST OF TABLES

Table 1: Cluster Model Tasks	73
Table 2: Standalone Message Time from the Driver Capture file in ms	80
Table 3: Standalone Message Time from the Server Capture File in ms	81
Table 4: Service Time for Standalone Model	82
Table 5: Cluster Message Time from the Driver Capture File in ms.....	83
Table 6: Cluster Message Time from the Server Capture File in ms.....	83
Table 7: Cluster Message Time from the Load Balancer Capture File in ms.....	84
Table 8: Service Time for Cluster Model	84
Table 9: Packet Sizes from the Standalone Environment Ethereal Traces.....	88
Table 10: Packet Sizes from the Cluster Environment Ethereal Traces	89
Table 11: Call Rate and CPU Utilization for Standalone Real System Measurements...	94
Table 12: Call Rate and CPU Utilization for Cluster Real System Measurements	95
Table 13: Comparison of Real System Data with Model Predicted Data.....	96
Table 14: CPU Utilization Comparison for the Standalone Environment.....	96
Table 15: CPU Utilization Comparison for the Cluster Environment.....	97
Table 16: Comparison of Real System Data with Calibrated Model Predicted Data....	107
Table 17: CPU Utilization Comparison for the Standalone Environment.....	108
Table 18: CPU Utilization Comparison for the Cluster Environment.....	109
Table 19: CPU Utilization Comparison for the Cluster Environment.....	114
Table 20: Maximum Throughput of Different Cluster Configurations	116

Glossary

B2BUA	Back-to-Back User Agent
CPS	Calls per Second
CSMA/CD	Carrier Sense Multiple Access Collision Detection
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSL	Digital Subscriber Line
GEIP	Gigabit Ethernet Interface Processor
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JSP	Java Server Pages
JVM	Java Virtual Machine
JVMPI	Java Virtual Machine Profiling Interface
LAN	Local Area Network
LQN	Layered Queuing Network
LQNS	Layered Queuing Network Solver
MGEN	Multi-Message Generator
MMUSIC	Multiparty Multimedia Session Control
NGN	Next Generation Network

PET	Performance Engineering Toolkit
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RFC	Request for Comments
RTP	Real-Time Protocol
SAME	Software Architecture and Model Extraction
SCTP	Stream Control Transport Protocol
SIP	Session Initiation Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transmission Layer Security
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Universal Resource Identifiers
URL	Uniform Resource Locators
UML	Unified Modeling Language
VoIP	Voice over IP
WAS	WebSphere Application Server
WML	Wireless Markup Language
XML	Extensible Markup Language

Chapter 1.0: Introduction

The telephone is one of the most commonly used technological devices for the past century and has been essential in our everyday lives. The public network which hosts this service has traditionally been known as the Public Switched Telephone Network (PSTN). This system has been tuned to give high performance and reliability for its users in everyday communication. As the communication technology moves towards to include the Internet technology, the traditional PSTN through copper wire is being replaced by the use of the Voice over IP (VoIP) technology. This new VoIP technology is classified as the Next Generation Network (NGN).

VoIP involves the use of network packets for sending communication data over the Internet and has the ability to perform all the same functions as the traditional PSTN. The most popular protocol used for setting a VoIP connection is the Session Initiation Protocol (SIP) [RFC 3261]. SIP provides a standard for setting up a method to transfer communication data over the Internet consisting of session connections and data transfers while supporting Quality of Service (QoS) [Wu03]. This process of packet communication involves sending packets in a specified order over the Internet in a form of SIP calls that are used to establish a connection between two users.

Many major telephone companies are spending resources on the research and development of SIP technology. They are planning to provide the traditional services in PSTN with the use of the new VoIP technology to consumers. Examples of such companies include Vonage and Skype. The main concerns the telephone companies are faced with when developing SIP includes performance, capability, reliability, and

resource management. This thesis looks into the performance aspect of a SIP implementation being developed at IBM.

1.1 Need for Performance on SIP

With SIP in its development stages, it is necessary to ensure this new telephony system can at least match the performance capabilities of the PSTN system in order to compete with the old technology. This new system also has to be reliable so the customers can accept and use this new system with confidence.

The traditional PSTN network has been developed and tuned for performance and reliability that the telephone switch experiences only a few seconds of downtime per year. The performance of PSTN has been thoroughly studied and modeled to characterize the system's behavior. Typically, in a commercial PSTN switch, the system should be able to handle 1 million to 2 million calls per hour. The reliability of this system should be higher than 0.99999 for switch availability while the expected call loss should be kept at the order of tens per million calls handled [Gurbani05].

The standard for performance and reliability is very high for PSTN systems. In order to move from PSTN to the new NGN such as VoIP, SIP has to match the same standard of performance and reliability the PSTN has set. In this thesis, performance tests were performed on a SIP system prototype that was being developed by IBM. Using the traces from these performance tests and an extraction method for the capture of files, a performance model was generated to predict the overall system throughput. The data was then compared between the predicted data and real system data to see whether

or not the measurements match up. Using the model analysis, suggestions and implementations were made in order to improve the overall system performance.

1.2 Related Work

Modeling is a very common practice as models provide a quick way to study a system before it is built. In the performance area, an important modeling tool is the Layered Queuing Network (LQN) model [Woodside05]. This tool is able to simulate the behavior of a system when a load is applied to it. In order to generate the model, the system must be carefully studied. There is a need to find a method to obtain the different performance values for the different modules in models in order for it to be solved.

A lot of research has been done on the performance model generation technique which poses an important aspect of performance analysis. One of the first techniques was proposed by Hrischuk [Hrischuk95] [Hrischuk95a] in the automatic generation of performance models using angio-traces. Angio-tracing is a method to capture the behavior of a system by recording the execution of the system through code instrumentation and taking the information gathered to generate a performance model.

Another technique for performance model generation proposed by Israr [Israr01] [Israr05] is called the Software Architecture and Model Extraction (SAME) technique. This technique uses generic traces generated from other programs to build the performance models by looking at trace patterns. By organizing and combining the trace data to interaction patterns where they represent forwarding, asynchronous, and synchronous interactions, a layered queuing performance model is created. This

technique allows the user to know the different patterns while building the performance models, thus more efficient than manually building the performance model manually.

Since SIP is a new technology to the market, there are not many published papers studying SIP's performance at the moment. A comprehensive and standard performance test plan is still needed for SIP. In regards to performance analysis and techniques, there is a lot of different varying work that is done in this area. The goal is to be able to use these techniques to measure the behavior of SIP.

Some simple performance modeling on SIP was done by Wu and Wang from National Central University (NCU) in Taiwan [Wu03]. They proposed a few methods for measuring the queuing delays with Markov Chains in different queuing models. Zhu from North Carolina State University (NCSSU) has also done some similar analysis for SIP in an IP Multimedia Subsystem (IMS) [Zhu03]; she has proposed and tested different scenarios to find bottlenecks in their SIP implementation.

There is some performance research that was done by Gurbani et al. at the Bell Laboratories and Lucent Technologies [Gurbani05] with their SIP implementation. They followed the standard test case from the SIPStone benchmark [Schulzrinne02] and used some simple queuing models and Markov Chains to analyze the performance of a SIP server system and also a system with proxy servers. In addition, other testing has been done under different configurations and the SIP server system's behavior was observed by looking at the response time of the calls.

De Marco, Iacovoni, and Barolli also performed testing similar to Zhu's, measuring SIP in their IMS system [DeMarco05] using a wired and wireless system to test the amount of packet loss at specific workload on the system. As well, they used a

closed queuing network to model with a finite state machine of the SIP protocol according to the SIP standard [RFC 3261]. Using their queuing model and test environment, different test runs were performed to test the amount of packet loss using a driver to send SIP Invite requests to the server.

This thesis studies the performance of SIP using the LQN performance modeling methods. From the above ideas in the related work, a simplified technique is used to construct the model automatically from the data obtained from a SIP performance test run. The following gives the contributions and outline of this thesis that describes the complete algorithm for this technique.

1.3 Contribution

My contributions from this thesis are listed as follows. First, performance models were built using a commonly available network monitoring traces such as Ethereal and a template created from a protocol call flow scenario. This is a non-intrusive way to measure the system with minimal instrumentation to the existing system. The algorithm first extracts the relevant information such as the time stamp and the SIP message from the Ethereal packets and stores them in a separate file in the form of packet trace. Using these packet traces, they are sorted according to the call flow ID to extract complete call flows from the traces. From the interaction patterns generated from the template, service times are calculated using the time stamps in the call flow traces. After several calculations with different call flows, an average was taken and was used as the service time for the model parameter.

The second contribution was that the performance model predictions generated by the LQN solver was able to be validated with the real system measurements with some model calibration. This shows that using the performance models is a valid way to do performance analysis and prediction for a telecommunication system such as the SIP implementation used in this thesis.

The final contribution was that bottleneck was found on this SIP implementation and improvements were made. By adding more server machines and sharing the network load with an extra port helps the system to handle more calls and utilize the SIP servers more fully. This is important for capacity planning and performance of the system as it uses more resources with the same amount of hardware.

1.4 Outline of the Thesis

This thesis is organized in the following manner. Chapter 2 describes the SIP architecture and how the system works. It also gives some information on the discipline of performance engineering and the tools that are used in the test runs on the SIP system. Chapter 3 describes the test scenario and environment setup used for the performance measurements of the SIP system, as well as elaborating upon the actual test runs performed on the system.

Chapter 4 depicts the construction of the performance models for the different environment setup and the steps took to obtain the performance numbers from the data for these models. With the performance numbers calculated, the construction of the model is complete and is ready to be solved and analyzed. Chapter 5 goes into details on

some techniques used to calibrate the system to ensure the predicted system data and the real system data matches. This chapter also suggests some ways to improve the performance of the overall system. Finally, chapter 6 describes the overall results and the contributions that have been made from this thesis.

Chapter 2.0: Background

The goal of this thesis is to study the performance of SIP. In order to accomplish this, there is a need to describe the SIP specification and architecture to model the performance of this protocol. There is also a need to study the ways to collect data for these models. This chapter begins by describing the architecture and functionalities of the Session Initiation Protocol (SIP) system that can be found in the Request for Comments (RFC) 3261 standard. After that, this section gives an introduction on performance engineering and modeling using different queuing modeling techniques such as QNAP and Layered Queuing Models (LQM). Finally, this section provides information about the open-source test tools that were used for running the performance capture tests such as nmon and Ethereal.

2.1 SIP Protocol and Architecture

The main Session Initiation Protocol (SIP) specification is published and explained in RFC 3261 (formerly RFC 2543) which is developed by the Internet Engineering Task Force (IETF). The specific group from IETF that worked on SIP is the Multiparty Multimedia Session Control (MMUSIC) working group, specializing in developing protocols that support Internet teleconferencing and multimedia communications.

The first draft of the SIP specification was submitted in 1997. Since then, many changes were made and a second version of draft was proposed in 1998. To make it

official, more changes were made and RFC 2543 for SIP was published in April 1999. After further revisions, another version which is known as RFC 3261 was accepted in 2000. This protocol specification obsoletes RFC 2543 and the RFC 3261 standard is used today by developers. The following section gives an overview of this specification for SIP.

2.1.1 SIP Architecture

SIP is an application-layer protocol that is used for starting and stopping a communication session between two or more participants. As stated in RFC 3261, it incorporates the use of other Internet protocols such as the Hyper Text Transport Protocol (HTTP) [RFC 2616] and the Simple Mail Transport Protocol (SMTP) [RFC 2821]. From HTTP, SIP makes use of the client-server design and the Uniform Resource Locators (URL)/Universal Resource Identifiers (URI) in its messages. With SMTP, SIP makes use of the text-encoding scheme and the header style in its own SIP packet headers.

The Internet Multimedia Protocol stack can be shown using a simple four-layer model from Figure 1. The lowest level, which is the Physical/Link Layer, can be a Local Area Network (LAN), a telephone line, Digital Subscriber Line (DSL), or wireless connection. This layer performs the sending of plain data over a common medium that is available between two communicating points. The next layer is the Internet Layer that uses the Internet Protocol (IP). It creates the headers for a string of packets that identifies the location of where it should be sent to and the routes packets take to reach its destination.

Figure 1: Simple Internet Multimedia Protocol Stack

The third layer of the network is the Transport Layer. The SIP implementation uses either TCP or UDP to set the method of transferring packets between the two parties. TCP use a reliable, connection-oriented transport to ensure the data was arrived at the destination successfully by using sequence numbers and acknowledgments. Packets are retransmitted until they are successfully received at the destination. The TCP client and server establish a connection using a 3-way handshake with SYN and ACK messages. When the connection is established, the two parties can begin to transmit data with the possibility of retransmission if the data segment does not give a positive reply after a set period of time.

In contrast, UDP provides unreliable transport controls over the Internet by using best-effort delivery. Since this is not a connection-oriented protocol, it does not use the acknowledgements or sequence numbers for data segments. This protocol does not provide error checking or error recovery so it is up to the higher layer protocols to provide this service. The advantage with this protocol is that it contains less overhead than the TCP protocol.

Other transport layer protocols that can be used include the Transmission Layer Security (TLS) [RFC 3436] and Stream Control Transport Protocol (SCTP) [RFC 2960]. TLS is based on the Secure Sockets layer (SSL) protocol [RFC 2246] and uses TCP for the transport. This protocol incorporates encryption with a one-time key to provide security during the data transfer. SCTP is similar to TCP but provides a stream-based transport instead of sending separate packets between the two communicating hosts. It has built-in message segmentation so that each message is separated at the transport layer to allow a more direct and reliable transfer. This allows a more efficient and reliable data transfer at the Transmission Layer.

The top layer in the network architecture is the Application Layer. This layer includes the SIP protocol and other media transport protocols such as Real-Time Protocol (RTP) [RFC 3550]. There is also the Session Description Protocol (SDP) [RFC 2327] that is above SIP that carries the SIP message. Along with the SIP protocol, it also uses other applications such as the Domain Name System (DNS) [RFC 1035] to resolve a symbolic name and Dynamic Host Configuration Protocol (DHCP) [RFC 2131] to obtain the IP information upon initialization.

All SIP messages are text-encoded so plain text describing the message is sent as it is over the wire. Some of the SIP fields use the URLs and URIs of the system to identify the location for the packet to be sent. As it can be seen, SIP works with different types of common applications and tools to perform the work. The following section describes SIP's functionalities in more detail and shows how SIP can be used for Internet communication.

2.1.2 SIP Protocol Overview

The Session Initiation Protocol can create different types of sessions such as Internet telephone calls and multimedia communications. This is usually implemented within an application that is used to perform these functions. Figure 2 illustrates a simple SIP call and the messages that are involved. The caller and the callee use the SIP phones to establish a session through the Internet.

Figure 2: Simple SIP Session Example

The SIP conversation starts when one participant sends an invitation, such as an INVITE message, to another participant requesting to establish a session. This invitation that is sent by the SIP client contains the session descriptions that allow the participants

to be aware of the type of session and the types of compatible media. SIP might also use proxy servers to route the request in between the two users to provide services between them. When the Invite request is received by the server, it processes the message and generates a suitable reply.

After the invitation request is processed, one possible response could be a ringing message that is sent back to the caller to notify that the invitation has been received and the alert has been sent. The information contained in this message is similar to the Invite message. When the phone call is accepted by the callee, an OK response is sent back to the caller to notify the start of the session. This message is also similar to the ringing message as described above.

The final step of the establishment for the media session is to send the “acknowledgement” request back to the callee from the caller to notify the start of the session transfer. From the messages, both the caller and callee already have all the necessary IP information and routing paths to start the transfer of the information in terms of the different SIP media. The session continues until one of the users initiates the end call procedures.

When one of the callers hangs up, a bye request is initiated between the caller and the callee. The initiator of this request is considered to be the SIP server while the other party is considered to be the SIP client. The initiator of the bye request waits for an OK reply acknowledging the end of the call. This completes one transaction using the SIP protocol for establishing a media session.

Other than establishing connection between the users, there are many other scenarios that need to be implemented for system. One important one is the registration

of the user to the registrar server. This scenario is important since this allows the server to be able to route the messages to the correct client. After the registration, the information of the user is stored in a database in the server that can be looked up when another party is trying to reach this user. This scenario is performed by using the Contact message to the server with all the required information. The server then replies with an OK message to acknowledge the registration.

Another scenario is when SIP is used in a presence and instant messaging server. Presence means the information of a particular person/device at that instant such as its status or activity and this information are being tracked continuously throughout the SIP session. SIP uses another type of message called the Subscribe request and the Notify message to send the required information. An OK message is also received when these Subscribe and Notify message are received.

There are many different scenarios that can be done using the SIP protocol as stated in the RFC. As described, all the scenarios are characterized by the different types of messages that are sent between the two parties. Each message can be categorized into different types that have different meanings and functionalities. The following section gives an overview of the different types of messages that are used for a SIP session and how they are used in different scenarios.

2.1.3 SIP Messages

There are different types of messages used by SIP to identify the different functions in the call flow. These messages, such as request and response messages,

define the status of the call for the connection call flow. An example of a request message includes an Invite message that initiate a call while the response messages is expected as a reply with the status of that request. Some of the common request and response messages are listed below:

- INVITE: a message containing the caller's and callee's information to initiate a connection between the user agents
- BYE: a message sent by a user agent to initiate the termination of the current session
- ACK: a message that is used to acknowledge the final responses for INVITE messages and other such as the 2xx, 3xx, 4xx that are described next and other similar class responses to finish the handshake procedure

There are six different classes of response messages that are used in the SIP call flows:

- 1xx messages: informational responses such as Trying and Ringing which indicates the call progress (e.g. 100 Trying – indicates that there are some pending actions that still need to performed to establish a call)
- 2xx messages: success responses indicating the request succeeded or are accepted (e.g. 200 OK – a message containing the media properties of the called party to indicate the request has been accepted)

- 3xx messages: redirection messages from the SIP server through different nodes on the network to establish a route for connection (e.g. 300 Multiple Choices – redirection response after the request has been sent to different proxy servers)
- 4xx messages: client error responses used by the server or user agent stating that the request cannot be fulfilled (e.g. 400 Bad Request – error message indicating the request was not understood by the server)
- 5xx messages: server errors where the request cannot be processed due to errors or problems at the server (e.g. 503 Service Unavailable – message is received stating that the service is temporarily unavailable and will retry after a certain amount of time)
- 6xx messages: global errors where the server knows that the request will fail wherever it tries (e.g. 606 Does Not Exist Anywhere – global error message indicating the user in the request cannot be found anywhere on the network)

The SIP call flows are based on these messages to establish a connection. In the following section, the SIP message header is described in more detail to understand how these messages work to communicate the information between the servers and the user agents. It gives the architecture and different header fields that are used in the SIP message to show how the information is being passed between the users.

2.1.4 SIP Header

After looking at the different types of messages, it is important to look at the actual messages to see how information is sent. All the different fields in the header are defined in RFC 3261. The SIP header fields follow the general format of the HTTP header fields. These fields are defined as “Header: Field” where the “Header” represents the type of information and the “Field” represents the value for that “Header”. Some of the common SIP Headers include: Alert-Info, Call-ID, Contact, Cseq, From, Max-Forwards, Retry-After, Subject, To, and Via. Here’s an overview for the header fields:

- **Alert-Info:** provides a “distinctive ring” option in the call
- **Call-ID:** a mandatory field that is used to identify the call between the two user agents using a unique ID number
- **Contact:** states the URI that identifies the resource requested or the request originator
- **Cseq:** a counter that is present in all SIP messages that increases for each request used by the SIP server to identify which message belongs to which call flow in case of multiple requests initiated by the same user agent
- **From:** contains the URI or a SIP-specified tag of the originator for the request to identify the users
- **Max-Forwards:** indicates the maximum number of hops that the message may take in order to reach the destination and will be decremented after each forward (SIP sends a 482 Too Many Hops error back to the sender if it exceeded the limit)

- **Retry-After:** approximates when the resource will be available again so it can continue to try to establish a connection
- **Subject:** indicates the subject of the media session for the call and can be used for screening or to decide whether to accept the call
- **To:** identifies the recipient of the request with the URI along with an optional tag
- **Via:** records the SIP routing path that was taken by the request by adding an entry to this field during each proxy forwarding point and is used to route the reply back to the sender

These are the main headers that are used to define SIP messages. The sample ethereal trace in Appendix A-1 shows how the SIP header fields look like. Studying the header fields helps us understand the architecture and sequence that are used in the SIP call flow.

2.2 Software Performance Engineering

In any developing system, measuring and analyzing the performance of the system is very important. There is a set of goals that needs to be achieved in terms of performance and timeliness of the system. Software performance engineering looks at the different ways and methods to analyze the data in order to improve the overall performance of a system [Smith02].

Performance is usually measured in terms of the response time and throughput of the system. The response time is defined as the time the system needs to generate a

response for a request or the amount of time needed to complete a scenario. The system throughput is the number of requests the system can process within a given period of time.

Software performance engineering contains a set of principles and approaches for creating responsive software to meet the performance objectives [Smith02]. It provides tools such as performance patterns and anti-patterns in the system software design along with other important performance principles to produce responsive systems. It also describes different techniques for gathering data for evaluation and the different types of analysis to be performed on a system. One of the main techniques is to build a model of a system to explore the characteristics in regards to performance. A model can predict the behavior of a system without having the system fully built and is able to suggest the areas where improvements are needed.

In order to effectively use modeling, different strategies for obtaining the most relevant data are needed. First, a very simple and quick abstract model can be implemented to identify the problems in the system early in the development. This is useful during the early design stages to define the bottlenecks in the architecture or test the different performance architecture of other alternative design solutions quickly.

The next strategy is to use the worst-case or best-case scenario of the measurement data to construct the model. This is necessary with estimated performance data since it gives the upper and lower bounds on the expected performance on the system. The best-case scenario can give the best possible throughput and see if the design is able to satisfy the performance requirements while the worst-case scenario shows the minimal capacity that the system is able to handle.

After using these models, the “adapt-to-precision” strategy can be used as development enters the later stages. At this time, the system has developed further so the performance data can be measured or estimated more precisely. The additional details can be used to expand the model to show additional information and behavior. These models can then be used to evaluate the system and identify any areas or system configuration that can be changed to improve the overall performance.

From the model analysis, problem areas are identified within the system. Software Performance Engineering supplies different solutions such as performance patterns and anti-patterns as described before [Smith00]. Performance patterns describe some of the best practice for developing responsive and scalable systems. Performance anti-patterns are common performance mistakes that slow the system down. The developers can use these patterns to correct the well-known performance related problems in the system. The following section provides more details on performance modeling and some of the modeling tools that are used for analysis.

2.2.1 Performance Modeling

In order to understand how to achieve the best performance results for an application, it is necessary to understand how the current system is implemented by examining the overall system behavior. In order to be effective, a prediction technique is needed early in the development stage to avoid major changes in the system after the development is complete. One method, as proposed by Smith as described above, is the use of performance modeling.

Performance modeling is important in terms of capacity planning to determine the amount of resources that are needed without actually building the system. It is used as a way to represent the modules' execution time, and how they affect each other's workload within the system, by capturing the main factors that determine the behavior of the system under execution. This can be used as a design mechanism to see how the developers can tune up the efficiency of the modules before full implementation. This is good practice since the system might be difficult to change after it has been fully implemented. It also saves development time and prevents performance failures while the system is developed. It can also be used during the other stages of the development process to compare different implementations to see how some of the additional features can affect the performance.

There has been quite a bit of research being done in this area of performance modeling with different approaches and frameworks. There are two different types of performance models: simulation models and analytical models. Simulation models are software programs that mimic the behavior of the system under test. These programs measure performance by counting the number of events and measure the amount of time spent at each component. The analytical models are based on mathematical calculations and computational algorithms to generate the performance parameters for the models. Some of these models that can be used include Markov Chains, Queuing Networks, and Stochastic Petri Nets.

One of the modeling tools is called QNAP and is a form of conventional queuing network which makes use of the queuing theories. With this tool, the tasks (or modules) in the system are lined up in a sequential order for the simulated user to be serviced.

Each module services the user for a time specified as the time needed to complete this task. While one user is being serviced, the other users who wish to enter are placed in a queue waiting until the module finishes its current task. During the simulation, there is a load of users generated by the input source that passes through the system. The tool then shows where the user queues are being saturated in order to find the bottleneck. The bottleneck is the module where the developer needs to improve the performance on.

Another way to represent the model is with the use of the Layered Queuing Network (LQN) models [Franks05] which is suitable for modeling layered software. A tutorial can be found for this tool written by Woodside and Franks [Woodside05]. It begins with the concepts of Markov chains and Petri Nets to capture the general case of a system. As the model grows, more interactions are added to the system and a layered approach is needed.

The conventional sequential queuing networks only capture customers visiting the hardware resources such as disks. These solvers are not able to capture the total behavior from different aspects and could potentially end up in infinite output error. LQN modeling captures both the hardware and software resources of the system and is well suited for client-server architecture.

The measurements that are used in the model's tasks are demand, service time, think time, and mean delay. This model also contains other external devices used in the system along with the tasks. The users, represented as "tokens" in the model, are then passed through the different modules in the system. These "tokens" are passed along through each of the modules in the order that they are inserted and are placed in a queue if the module is busy servicing another "token". Each task in the model serves as a client

where it can call other tasks for their service or access some external devices. While it is calling the other task, the current client is blocked until the called tasks are finished. This is classified as a rendezvous call. This is shown when the performance model is simulated with the Layered Queuing Network Solver (LQNS).

After the model is solved by the LQNS, an output file containing the performance values of the system is generated. These performance values includes the following: mean throughputs of the tasks, entries, and processes; mean wait time for each entry; mean service times of entries; and the mean utilization of tasks, entries, and processes. These values indicate how the system can handle the stress while there are many users using it.

The modeling tools are used for capacity planning and for locating performance problems of the system. From the LQNS output file, it is possible to identify which resources are needed in order to increase the overall maximum throughput. This thesis uses the LQN modeling technique to perform these tasks on the SIP system.

2.2.2 Ethernet Modeling and Simulation

Within performance modeling, it is necessary to look at Ethernet modeling as the queuing models require a way to include the Ethernet resources used in the system. Since the behavior of the Ethernet is load dependent, it is not represented well using the conventional queuing network models. In a performance model where the modules are connected using Ethernet, it is important to understand the behavior to see how it affects the overall performance.

The IEEE 802.3 standard [IEEE] defines the structure of a frame for basic network communication. The Carrier Sense Multiple Access Collision Detection (CSMA/CD) protocol is the method of controlling these frames going through the network to the destination host [Cisco05]. Carrier sense means that each station continuously listens for traffic. Multiple Access means stations can transmit data as soon as the network is available, and Collision Detection is a method of improving CSMA by ending the transmission as soon as collision is detected to reduce additional collisions on retransmissions. Collisions occur when two stations that wish to transmit packets attempt to access the network at the same time. The two electrical signals on the network at the same time are what cause collisions to occur. The station that first detects a collision sends out a jamming signal to alert other stations of the collision. Stations involved in the collision each initiates a “back off” algorithm that randomly generates the amount of time that station must wait before attempting to access the network again.

In performance modeling, it is important to study the behavior of the system under a heavy load. When Ethernet is a part of the system, there is a need to study the behavior of the Ethernet when network traffic is high due to contention delays. At ideal conditions, Ethernet contention delays can be easily calculated as shown in [Lazowska84] and section 4.4. When traffic is high, there are increased collisions, packet loss, and retransmissions that need to be taken into consideration. This has an effect on the results in the calculation for the Ethernet service time.

To simulate the workload of the CSMA/CD protocol at a high traffic volume is a very complicated and cumbersome task. There are many different attempts to model and simulate the Ethernet but there is still not a method that can clearly model the behavior.

Numerous researches have been done but they contain lots of simplifying assumptions. These approaches are all event driven where the movements of packets are expressed in terms of events.

Generally, there are two approaches to simulate this behavior: centralized approach and distributed approach. Using a centralized approach to simulate the Ethernet, the medium is simulated by an active entity that keeps track of the packet in each of the machines that are part of the network. This entity then informs the other machines or stations about the state of the medium. This medium also detects and computes the time a collision occurs and sends out a jamming signal. These stations only simulate the packet transmission and retransmission as collision and packet drop are due to buffer overflow. This method is able to simulate all the different aspect of the CSMA/CD protocol accurately, but the implementation is extremely complex.

Another way, proposed by Wang and Keshav, is to use a distributed simulation approach [Wang99]. The first step is to build a detailed, yet simple, simulation of the CSMA/CD protocol using a network simulator to reproduce a variety of workloads with different configurations. With this simulator, a large number of empirical performance measurements were generated based on a large number of configurations. All the measurements are then compacted together into a simpler and general model. This model then reflects the dependencies of the link throughput and packet delay with the different loads, packet sizes, and buffer sizes.

The third step to this approach is to use this performance model to develop a “Fast Ethernet Simulation”. The purpose of this simulation is to perform a performance prediction based on past empirical measurements. Using the different techniques

described by Wang and Keshav for traffic monitoring and performance prediction, they were able to find a way to quickly gather performance data for the Ethernet system that was up. With this “Fast Ethernet Simulation”, the results can then be verified with the data obtained from the detailed simulation in the first step.

Ethernet simulation is a very difficult task and there has not been any simulation that has successfully been able to fully model the behavior of Ethernet. At ideal condition with a low throughput, it is simple to calculate and simulate the packet delay for the packets. At a heavy workload, this is a much difficult task due to the retransmission and possibility of packet loss as stated in the CSMA/CD protocol. It is known that at a heavier load, the performance at an overall communication system can be affected at the Ethernet level. There is still a big need to find a way to accurately model and simulate the CSMA/CD protocol. This is a broad area that still needs to be researched.

2.3 SIP Server Test Components

In developing a SIP implementation and testing the system, different tools and components are needed to generate traffic and to measure results. This section describes the different components that are involved for deploying and testing the SIP system while the following section describes the tools used to measure the system’s responses.

2.3.1 SIPP Client

The SIPP client [Gayraud05] is an open source tool that generates traffic for the SIP protocol used specifically for performance testing [SIPP]. This tool uses the INVITE and BYE methods from SIP to establish multiple call sessions and then releases them. This client has built-in user agent scenarios as described by SIPstone such as the User Agent Client (UAC) and User Agent Server (UAS), but it can also use custom Extensible Markup Language (XML) scenario files for different types of call flows. The UAC and UAS are the standard scenarios for a SIP call where UAC initiates the call and the UAS generates the response. These scenarios are described in Section 3.1. As the SIPP client runs the test, it shows a real-time display of statistics for the messages that are being passed through the client. The tester also has the capability to adjust the call rate dynamically in the middle of a test run.

While running a test, the user can inspect the output display to see if the messages are able to complete the call flow. If there are unsuccessful calls, the client displays them as “Unexpected-Msg” or “Timeout”. The cause of this error could be that the server does not have the resources to complete the call transaction in time. This tells the user the maximum number of calls the server can achieve.

SIPP is suitable as a test driver in different SIP environments. It can be used with different types of servers: SIP proxies, back-to-back user agents, SIP media servers, SIP gateways, and other types of servers. The SIPP client can also simulate the UAC/UAS call flow without the use of a SIP stack as the UAS incorporates the basic functions of a SIP stack. This can be used to test the capabilities of a system without the full

implementation of a SIP server. It is also a quick way to make sure the environment is capable of handling SIP messages.

The SIPP client's main function is to generate a load for SIP performance testing based on SIPstone's UAC scenario that generates the calls for the test. Since there are many different scenarios that can be tested, there is a need to customize each test run. SIPP has a capability of using user-defined XML scenarios to run as part of the test which emulates different non-standard scenarios. In the XML file, the user can define the messages to be sent and their expected replies, the length of a pause delay, emulation of packet lost, and specification of the intervals for response time and call length. For the messages sent, the XML file can include all the header fields that are included in the message.

Generally, the results of the behavior that are being studied include the maximum call rate the system can handle along with the response time of the message. It is also important to make sure all the calls are successful with minimal retransmission. This data can then be further analyzed to improve the overall performance of the system.

2.3.2 IBM WebSphere Application Server (WAS)

The SIP implementation requires a middleware to deploy the SIP stack and the application that is associated with the server which, for this thesis is a platform called WebSphere Application Server (WAS) [WAS06]. This application server contains frameworks and components based on the Java 2 Enterprise Edition (J2EE) technology. This application server is essential as it contains some common Java functionalities that

are used during development and also allows easy deployment of the SIP server. This allows the developers to save time and money in the development of the enterprise systems.

J2EE has all the features in the standalone Java development environment to deploy Java standalone applications plus the ability to deploy enterprise systems such as web services. There are a few components in the J2EE technology: standalone clients, applets, web components, and server components. These standalone clients are simple applications that are written in Java and run within their own Java Virtual Machine (JVM). Usually, these client applications are compatible with the Java 2 Standard Edition (J2SE) platform. This allows the clients to use the J2SE runtime environment to run the client. Applets are like the application clients but they are executed within the web browser. Since most browsers have built-in JVMs, they have the ability to host and execute these applets. The web components are server-side components used to provide the interface to return to the client. These include the use of Java Server Pages (JSPs) and Java servlets.

IBM's WAS is a middleware solution that integrates all the standard J2EE features along with other IBM developed modules that can be used in different areas. This middleware solution is available for different operating systems and different system architectures. More information can be found on their website [WAS06]. An important feature of WAS is the ability to create a cluster of servers with different machines. There are two different versions of WAS that are available: Application Server and Deployment Manager. As described from above, the application server is used to deploy the Enterprise servers that were developed while the deployment manager is an

administrative server that manages all the different application servers to be used as part of the cluster.

A cluster is defined as a set of machines that are configured closely together that can share the workload within a system. In a WAS cluster, the application servers work together to handle the workload evenly. The role of the deployment manager is to manage all configurations that are set for the system. It defines the roles of the servers for different aspects of the system such as assuming the role of a proxy server or a load server. With WAS, any machine in the cluster acting as a server can be treated as one unit to handle the various service requests. In this thesis, the SIP stack server along with the application server is installed and deployed on an older version of WAS 5.102 and all the testing is done using this application server while the cluster is managed by a deployment manager.

2.3.3 Back-to-Back User Agent

The back-to-back user agent (B2BUA), as defined in the RFC 3261, is a program that is used between the IMS (IP Multimedia Subsystem) and the external network. This agent can process SIP INVITE messages similar to the UAS and acts as a UAC to initiate outbound calls. This application is installed in an application server such as WAS and is used to handle all the SIP messages that are coming to the system.

This agent acts as an interface that connects the SIP stack code to the server for the SIP communication and allows the SIP server to service the request. The main goal is to take the different in-coming SIP calls, modifies them, and then sends them in another

form to the destination such as the receiver's phone. Its functionality can provide centralized call management, VoIP between different LANs and WANs, and cloaking endpoint locations.

The basic design issue of the B2BUA is how it handles different types of SIP messages. It reads in the request and extracts the information from the SIP message header. This information includes the command, message ID, sender and receiver URI/ports, and other identification parameters. The B2BUA processes the request and sends the correct message type to the destination specified from the information.

The B2BUA can employ other SIP services, but this can also become a single point of failure of the system. This potentially makes the service less reliable. There is also an increase of latency, possibility of packet loss, and it is also possible to break the end-to-end nature of an Internet Protocol.

The most common form of B2BUA is an application layer gateway (ALG). The ALG can be integrated in the firewall so that the traffic for SIP can enter into the system. The B2BUA is a common way to serve as the SIP server to handle the SIP calls. In a fully deployed SIP server system, B2BUA is the main application that is used to customize the server's behavior.

2.4 Performance Tools

While running the SIP system, other external tools are needed to run in order to capture the behavior of the system. Some basic performance tools include a sampling profiler that samples the attribute of the system at different intervals. These tools

generate traces that can be used for further analysis. This section gives a brief description of the different performance tools that are used for the SIP testing in this thesis.

2.4.1 Performance Profiling Tools

At IBM, the developers have developed some performance capturing tools for performance testing. This section gives an overview about a profiling tool called “tprof” [TPROF], also known as the trace profiler, which is part of a tool suite called Performance Inspector [Mata04]. This toolset is used to identify the different performance problems and characteristics. Some other tools included in this suite are Above Idle, Per-thread Time, System Trace Data Collection, Jprof, and Java Lock Monitor.

Profiling can be defined as the process or the ability to monitor and trace different events while the program is running so that the user can track the cost of these events. It involves analyzing the pattern of resource use from that particular program. Profiling is able to show the processor time along with the percentage of program execution time used for each of the processes running in the program. These traces are the basis for the calculation of performance values used in performance model generation. Generating traces is usually the most complicated part of generating performance models since it is difficult to obtain reliable values without implementing the system. Sometimes, developers have to estimate these values if there is insufficient information given for that particular module.

The next step to generating a model is to use the traces generated to obtain the information on how the different modules and threads interact with each other. It is important to see how the methods are invoked and which components are involved. Each of the tasks recorded in the trace should have a starting time and an end time. This can show the amount of time it required the task to execute; thus, it is important that the traces include a timestamp from the execution start to end for calculations. From the traces of the system the profiler is executing on, the user can analyze them to observe any patterns to obtain a clear picture of how the entire system works as a whole.

One way to capture the traces from the program is the use of code instrumentation. Profiler code instrumentation consists of adding extra code into the server programs or the developed system to gather the information about the program's execution. For example, IBM's tprof requires adding code (or "hooks") into the Linux kernel in order to generate the performance data needed for the profiler.

IBM's tprof is a profiler that collects information on which part or module of the code is executing during set time sampling intervals. For example, tprof gathers data through a system-clock interrupt at a rate of 100 times a second. When the interrupt happens, the kernel "hooks" provides the address of the instruction that the system is executing. This information is then stored in the tprof trace buffers. After the test run, the tool generates a summary report of how many ticks each instructions gets and groups them according to the process, thread, module, and subroutine. Using this, it is possible to see where the code is being executed the most inside the system and allows users to find where the majority of the work is being performed.

Along with tprof, another profiler called jprof is also included in the Performance Inspector test suite. Jprof, also known as the Java profiler, is a profiling agent for Java that dynamically responds to the Java Virtual Machine Profiling Interface (JVMPI) [Suns03] events during runtime. It also provides JIT address-to-name resolution to support data reduction as done for tprof. This profiler can be used where the JVMPI is available at the JVM that is installed on the system. The output generated gives lots of information regarding the hotspots that are located in the JVM while the test is running.

Jprof can be very useful; however, it uses resources extensively and can cause a lot of overhead during test runs. JVM requires a lot of resources to perform all the JVMPI functions with the Java application to perform the captures. This means that jprof should not be run during a performance test run but during a normal test run to understand the system. This way the profiler and the system has enough resources to handle the service requests to generate meaningful profiling data.

Profilers are useful in identifying the bottleneck within the system's code. Different profilers concentrate on looking at different areas of the system and identify which area the system is spending the most time on. While doing profiling, it is important to choose the appropriate profiler to use to obtain the relevant results. Profiling is a good way to understand the behavior of the system at a lower level.

2.4.2 System Monitoring

Other than profiling with tprof, there is another tool called "nmon" [Atkins05] that IBM has developed to handle the performance capturing and analysis. This tool is

developed internally for IBM's use but is available for other developers in the open source community from IBM's Developer's Works. This tool supports the AIX and Linux operating systems.

Nmon, also known as Nigel's Monitor, is developed by Nigel Griffiths and is used to monitor and analyze many different performance data: CPU utilization, memory use, kernel statistics, disks transfer, free space on file system, and many other performance related information. The concept of this tool is similar to the profiler described above except no instrumentation is required to run this tool. It has the ability to display the performance data on the screen and to update it dynamically or it can output all the performance data into spreadsheet. This file can be used for later analysis and generate graphs for performance reports.

As mentioned above, nmon has a GUI mode and a background mode where it writes data to a file. In the GUI mode, the user can choose which performance data to follow by selecting different options. Nmon updates the information in a bar graph format onto the screen as the test is running. In the background mode, the user can specify how often the tool should take a snapshot of the system and how many snapshots to take before stopping. Every snapshot is recorded in the output file.

On top of nmon, another IBM developer has created a program called nmon analyzer [Griffiths05] that organizes the nmon data to produce readable graphs automatically. Nmon analyzer is in a form of a spreadsheet where it loads up and parses through the nmon data. It eliminates all the inconsistent data and produces graphs for each major section. It also calculates other data such as weighted averages for hot-spot

analysis, distribution of CPU utilization by processor, and many others. This tool is useful for users to analyze the nmon data obtained from the performance run.

2.4.3 LAN Analyzer

In order to capture the network call flow behavior, a tool is required to capture the packet being transmitted over the system. One of the popular network analysis tools is called Ethereal. This tool is an open-source project that is used to analyze protocols on UNIX and Windows. It has the ability to capture the network traffic on the system that it is installed on.

This tool can be started in two ways: Ethereal (GUI version) and Tethereal (command line version). Both versions work similarly and the results generated are the same. The command line version is useful with automation since it can start and stop with just a command-line option. The capture starts by listening on the wire where all the traffic is coming through. This tool supports many different types of network and listens on different protocols. The user can identify which types of packets to filter on during the analysis with Ethereal after the capture file is obtained. After the capture test, it generates a capture file where all the packets are recorded for further analysis. Each packet contains the header file which has all the parameters and values for each of the header fields.

Using this tool for capturing SIP messages is very appropriate. It displays various amounts of different information at varying protocol layers including all SIP headers for each packet captured. At each layer, it displays all the information that is used to

describe the packet including the source/destination hosts, ports, protocol used, and time.

All these information are important when analyzing a protocol.

Chapter 3.0: SIP Performance Benchmarking

In this thesis, SIP performance tests are performed on a SIP implementation under development at IBM. Studying its behavior is important so that its capacity can be increased as the system gets developed. In order to do this, it is necessary to measure and model the SIP system. This section describes the configurations of the SIP system under a load test. The system was benchmarked using two different configurations: standalone environment and cluster environment. The data gathered during the runs was used to parameterize the performance models described in Chapter 4, and used to compare the accuracy of the predictions of the model.

3.1 *SIP Performance Benchmark: SIPstone*

In the study of SIP performance, there is a need for some basic guidance and direction for the testing. Currently, there is a published benchmark that can be used as a guideline for a set of different recommended test scenario settings and performance measurements. For SIP servers testing, the SIPstone document [Schulzrinne02], written by a group from Columbia University, proposes a simple set of benchmarks for evaluating the performance of the proxy, redirect, and registrar servers. These tests were designed to be simple so that they can be easily recreated. The workload generated by this benchmark includes registrations, successful forwarding, and unsuccessful calls.

This only measures the server's performance, but not the server's protocol compliance or robustness, nor do the tests evaluate the capabilities of the system.

The architecture proposed by SIPstone, known as "server under test", consists of:

1) a SIP proxy, redirect or registrar server whose performance is being evaluated, 2) a load generator which creates SIP requests, 3) a call handler that simulates a user agent server, and 4) a central benchmark manager which controls the workload. These modules can be connected through any network.

In regards to performance, the metric used is the number of requests per second a system can handle with a certain success rate and minimal delay. The delay is important in the testing since it is necessary to avoid the retransmission of packets. Some issues that can affect the tests as stated in the benchmark includes the architecture and size of the user lookup database, retransmission of packets at the upper transport levels, the use of TCP or UDP, and other non-SIP related traffic. These have to be addressed in the test cases and be kept consistent during the test.

The SIP requests are generated by the user agent client (UAC) with a call flow as shown in Figure 3. There can be more than one UAC used during a test. The UAC talks to the server and generates all the messages that are required for the caller. The information needed at the UAC include the inter-arrival time for the requests for each host, the type of request, transport protocol, and the number of requests to generate.

Figure 3: User Agent Client Scenario

The load generated by the UAC is sent to the call handler called the user agent server (UAS). The UAS receives the INVITE request from the UAC and answers with a “180 Ringing” message and “200 OK” message. The UAS also has the ability to act as a redirect point for a call. This UAS acts as a SIP server where it handles the requests sent by the UAC. Figure 4 shows the diagram of the UAS scenario call flow where it is similar to the UAC scenario.

Figure 4: User Agent Server Scenario

This SIPStone benchmark presents 5 different tests: registration, outbound proxy, redirect, proxy 480, and proxy 200. All these tests are done using both UDP and TCP protocol in a series of test runs at different loads. These tests must be at least five minutes long since the system needs to enough data and able to reach a steady state to obtain an accurate response time for the system.

3.2 Test Environment Setup

All the testing done in this thesis was performed at the IBM Software lab in the North Carolina Research Triangle Park site. This was a performance project that was being worked by the Pervasive Computing performance group on the SIP implementation that IBM was developing during 2005. This section gives a description of the

environment setup of one of the prototypes that was used during one of the testing cycles and was used for this thesis.

3.2.1 Hardware Configuration and Setup

In the SIP environment, there are two different sets of machines: driver machines and server machines. The driver machines contain the code for the load driver while the server machines contain the code for the load balancer or the SIP application. For the standalone configuration, only the SIP application server is present.

Both driver and server machines are IBM blade servers built for performance and are necessary to obtain consistent results for the tests. These machines are all identical models to ensure easy scalability. The system was developed to have the blade servers host all the information such as server code, testing tools, etc. Each set of the machines has its own IBM blade server chassis. These chassis are infrastructures that help manage all the blade servers that are inside it and are easy to install.

The IBM blade server chassis (model #: 8677) used is for the HS20 blade servers. The purpose of a blade server chassis is to hold many different high performance blade machines in a compact space while sharing some basic network and hardware peripherals. These blade server chassis also provide redundancy for the blade servers and a central management console to perform administrative work on the blades.

The blade chassis contains integrated dual Gigabit Ethernet connections for high speed transmission of data between the blades in the chassis. This can be treated as a virtual network for the machines inside the blade chassis. For connection to outside

traffic, there is a Gigabit switch module to connect to a Local Area Network. This connection may be through a firewall or filter to regulate the traffic going in and out of the chassis blade servers.

The driver machines and the server machines are on different chassis and these chassis are connected through a network using a gigabit switch. This setting emulates a real-life network as the servers are all located in one location while the users are sending requests from outside the server environment. It is also important to use a gigabit switch to ensure minimal network congestion between the switches and machines. The network properties of the Ethernet network used for testing was verified before running the tests to make sure that the network is capable of running at maximum capacity. Data transferred between the blade servers within the same chassis does not go through the external network; rather, the chassis handles the data transfer itself. This is similar to a harddrive to harddrive transfer but it is still considered being part of the overall Ethernet environment. Using the chassis, it shows that there is no big noticeable transfer delay between the blade-to-blade transfers of data. Figure 5 shows the overview layout and connection for the chassis.

Figure 5: Test Environment Setup

All the machines on both the server chassis and driver chassis are running on Red Hat Enterprise Linux 3 Enterprise Server (RHEL3 ES). The kernel version used is 2.4.32 along with all the updates on all other tools in the system. All the system modules are set to use the best and most updated drivers. A test was also done between the machines using Multi-message Generator (MGEN) [MGEN] created by the Protean Research Group to ensure the network connections between the machines are suitable for testing. Network connection and basic network throughput is important as the source of the bottleneck should not come from the network.

The configurations of all machines are the same so the performance values are expected to all be identical among all the machines with the same role. There was some tuning such as memory allocation done on the Linux kernel to ensure optimal operating system performance. There was also tuning performed on WAS to ensure the JVM and the application server are using the most efficient methods to handle all the requests.

For data analysis, a few tools are required to capture the behavior of the system. The most important tool is Ethereal for capturing network packets. This is used to ensure that the call flow is correct during testing. Some profiling was done also using the tprof profiler to study the CPU utilization during execution. For the overall performance capture, IBM's nmon tool and analyzer was used to look at the behavior of each individual machine as described in Section 2.4.2. All the test tools and capturing tools needed were installed on all machines and were used during the tests.

The setup for the standalone and the cluster environment are a little different. As stated from the standalone SIP Invite scenario, this standalone environment contains only

the driver machine located in one chassis and the SIP server machine is mounted in the other chassis. The simple deployment diagram in Figure 6 shows the configuration of the standalone environment.

Figure 6: Deployment Diagram for Standalone Environment

For a clustered environment, the load balancer machine is mounted between the driver machine and the server machine. The driver machine is located on one chassis and the load balancer and the SIP server is located on the other chassis. The reason for putting the load balancer and the SIP server on the same chassis is that they represent the SIP cluster under the WAS server. The users send SIP requests to this cluster and all the work is done on this chassis to handle the requests. Figure 7 gives an overview of the cluster environment.

Figure 7: Deployment Diagram for Cluster Environment

3.2.2 Load Driver and SIP Server Software

The driver code that is used in the driver machines is a simple proprietary inviter program written in 'C' that performs the SIP Invite function and ends the scenario when it receives the SIP OK message. This program is similar to the SIPP client but it concentrates only on this specific Invite scenario described in section 3.3 and contains its own call tracking. This driver creates the SIP INVITE message with the necessary SIP header format following the SIP standard to send to the destination specified in the driver's parameters. There is also a SIP redirect program that listens for a specific SIP message on a specified port and sends the appropriate SIP message back to the server through the specified port. These programs are created specifically to simulate the call flow as described in the invite scenario.

The driver is meant to generate the traffic for the server with minimal overhead so there is minimal impact on the overall performance of the system. The only objective for the driver is to generate the INVITE message while the SIP redirect program only

redirects the message after receiving the INVITE message from the server on the port it is listening on. During the test, one instance of the inviter is running while three instances of SIP redirect are listening on ports 5076, 5074, and 5073. It is important to have all these instances of the inviter and SIP redirect running in order to complete the call flow.

The inviter and the sip redirect program acts in the role of the SIPP client in the testing. The roles of these programs can be mimicked with the SIPP clients using appropriate XML input files. The reason that the inviter and redirectors were used is that they create the least overhead since they only perform the invite scenario, and they generate specific output that is easy for analysis. Other than that, their purpose and behavior are the same of the SIPP client and are used in this thesis.

For the SIP server blade servers, they contain the code of the SIP stack that handles all the SIP functionalities. This SIP stack is implemented as Java code that is embedded with the Application Server. On these blade servers, they have been installed with WebSphere Application Server (WAS) version 5.102 along with the SIP stack code to be used for SIP testing. This server is capable of doing all the required functionalities as stated in the SIP protocol and the SIP stack code is fully implemented.

For the enterprise application, there is a simple proxy redirect program that handles the SIP message redirect on the server. This enterprise application is an interface installed on top of the SIP stack and uses the SIP stack to perform the necessary message forwarding to the SIP stack to complete the invite scenario. The concept of this enterprise application is similar to the B2BUA, but this redirect application only performs the message redirections used in this scenario. This application contributes minimal overhead to the system to ensure accurate performance measurements on the SIP stack.

3.3 SIP Invite Scenarios

From the SIPstone's list of scenarios, one of the most important scenarios is the SIP Invite where the two parties are trying to establish a connection by sending an INVITE request. All the testing in this thesis is based on a modified version of this SIP Invite scenario because this function is required as the first step to establish a connection between the user agents. All other scenarios require the establishment of connection before they can occur. The sections that follow describe the scenarios for these configurations: standalone, cluster, and the scenario where the exception occurs. A summary of Ethereal trace of a call flow for a simple Invite scenario can be found in the Appendix A-2.

3.3.1 Standalone Invite Scenario

The standalone Invite scenario contains the generic Invite request messages along with all the response messages necessary to ensure that a correct connection is established. The scenario ends as soon as an "OK" is received from both sides stating that they are aware of the connection and ready to receive data. There are also two redirection hops using the 300x message to act as hops from different proxy servers to establish a route between a user and the server. The following use case contains the call flow of a standalone SIP system containing one driver machine and one SIP server machine. Figure 8 contains the call flow of the scenario based on the SIP protocol and the proposed setup of the system.

SIP Call Flow

Figure 8: SIP Invite Scenario

1. The driver sends an INVITE message from port 5071 to the container machine which is listening on port 5080
2. The Container replies with a 100 Trying message from port 5080 to the driver who listens for this message on port 5071.
3. The Container sends an INVITE message from port 5080 to the driver machine where SIP redirector 1 is listening on port 5076.
4. The SIP redirector 1 on the driver machine sends a 300 Multiple choices message from port 5076 to port 5080 on the SIP container machine.

5. The Container replies with an ACK from port 5080 back to SIP redirector 1 who is listening on port 5076.
6. The Container sends an INVITE message from port 5080 to the driver machine where SIP redirector 2 is listening on port 5074.
7. The SIP redirector 2 on the driver machine sends a 300 Multiple choices message from port 5074 to port 5080 on the SIP container machine.
8. The Container replies with an ACK from port 5080 back to SIP redirector 2 who is listening on port 5074.
9. The Container sends an INVITE message from port 5080 to the driver machine where SIP redirector 3 is listening on port 5073.
10. The SIP redirector 3 on the driver machine sends a 200 OK message from port 5073 to port 5080 on the SIP container machine.
11. The Container sends a final 200 OK message from port 5080 to port 5071 on the driver machine. When the driver receives this message, it knows that the connection has been established and the scenario is complete.

NOTE: The container is listening on port 5080 instead of the standard port 5060 since the SIP system is designed for the messages to enter the load balancer using port 5060 and then forwards to the container using a manually set port of 5080. This is shown in the cluster environment but the standalone environment skips this port. In a standalone environment, the driver sends the messages directly to the container using port 5080.

For our performance testing purposes, only the server requires performance analysis; therefore, it is practical to concentrate on the server's behavior on the Ethernet. Since all other components from the driver are not important in the analysis, they can be simplified into a "black box" as the driver machine. Please refer to figure 9 for the standalone call flow diagram.

Standalone SIP Call Flow

Figure 9: SIP Standalone Invite Call Flow

3.3.2 Cluster Invite Scenario

In the Cluster environment, the main call flow is exactly the same as the standalone environment in Figure 9, except that the driver sends the message to the standard SIP port 5060 on the load balancer machine instead of port 5080. The load balancer then forwards the messages to whichever SIP container server the load balancing algorithm chooses through port 5080 on both sides. The SIP containers then replies to the load balancer who then directs the response back to the driver.

Figure 10 shows the message flow sequence diagram of the SIP Invite scenario when using the load balancer as an intermediate forwarding node. The overall call flow is the same; however, the network traffic is increased. With the extra network traffic, the behavior of the system should be different from the standalone environment. This SIP Invite scenario used is also the baseline test with one load balancer and one SIP server. More machines can be added as part of the cluster, but the call flow remains the same. This is discussed in Section 5.6 where it discusses capacity planning for the SIP cluster configuration.

Cluster SIP Call Flow

Figure 10: SIP Cluster Invite Call Flow

3.3.3 Invite Scenario Exceptions

During ideal conditions, the Invite scenarios are executed as described in the call flow above. For performance testing, the conditions may be different because the test runs are under a heavy load that uses most of the system's resources. While the system is under load, the call flow scenario may behave differently due to the time requirements for the messages to be received and the contention with the resources and network. The two main problems or exceptions that can happen in this situation are message retransmission and message loss. This section explains the call flow when the system is not behaving as intended.

In order to ensure the completion of the call flow as stated in the SIP standard, the system issues a retransmit of the same message if the reply from the original message is not received in a timely fashion. This retransmission message is sent in the case that the original message is lost. While this is good for the reliability of the system, it creates extra overhead for the system performance which could impact the overall system behavior. It is important to study this behavior since reliability usually costs performance in the design.

When the system and network are saturated to the point that they cannot supply a response to a message within a reasonable time, these calls are classified as unsuccessful, and a "Never Got Response" message is sent to the output screen. The call is then terminated and this error message is sent back to the caller. In terms of performance testing, this is a major error since the system is not able to handle the incoming call load

due to the system being fully utilized. In performance testing, it is important to discover the limit where all the calls are being completed successfully.

An example of retransmission and failure messages from a test run is shown in Appendix A-3. These packets are captured when the system is overloaded and causing the system not to be able to respond to the user quickly enough. There can also be collisions with other packets that attempt to access the network at the same time. This is an example of a “traffic jam” on the system’s network. At a light load, this might not be a problem, but this is a serious problem when a heavy load is generated.

It is important to study these exceptions in the system and how they appear as they have tremendous effect on the overall system. To measure the system throughput, it only takes note of all the successful calls. In order to improve the overall performance, it is necessary to eliminate the retransmission and packet loss as much as possible. With the understanding of the SIP call flow, Section 3.5 describes the actual test setup that was used for this thesis.

3.4 Performance Engineering Toolkit (PET)

To analyze performance test results for the different systems at IBM, there is a need for a framework that can handle the gathered data to make it available for users. On the IBM performance team, Curtis Hrischuk developed a toolset that can be used for different projects to organize the performance data. This tool called the Performance Engineering Toolkit (PET) takes in a specific set of test run data to get analyzed and

organized. With this sorted data, different graphs are generated for the user to show the behavior of the system.

PET is an internal tool used only by the IBM Pervasive Computing Performance Team to analyze the data from different types of test runs. This tool is mostly written in Perl to analyze a specified set of collected data entered into the tool. This tool contains two main Perl scripts to perform the analysis. At the command line, the user can specify the time interval for the analysis so that it displays only the information needed from certain time frames in the graphs. There are also other different options that can be set with PET to ensure the relevant information is being processed.

The first Perl script performs the data analysis on the measurement files by taking the collected data to generate a set of files with organized data called “PerfColumns”. These PerfColumns are comma separated files (.csv) containing measurement values such as the system utilization for a specified time range. There are also different types of PerfColumns for different analysis. At the time interval between data points, an average of values within range can be taken and stored as the Average PerfColumn. There is also a Summary PerfColumn where some values such as the minimum value, average over time, average over count, maximum value, standard deviation, sum by duration, and other statistics are calculated from performance data. These PerfColumns can be easily exported to Excel for further processing.

The second script uses the PerfColumn data from the first script and constructs a plot that can be viewed by an open-source program called Gnuplot. A different graph is generated for each type of data from the PerfColumn. Figure 11 is an example of a PET graph showing the CPU utilization for one of the test run.

Figure 11: Sample PET Graph Showing CPU Utilization

The graph above shows the system's CPU utilization over a range of time for a set of three different machines. The x-axis is the time over the time interval specified. The y-axis represents the value for the CPU utilization for that period of time. The legend contains the different machines that are plotted on the graph and identifies each one of them with a different data plot.

PET is able to take in certain types of measurement data to be analyzed: VmStat, NetStat, IoStat, NicStat, nmon, SIPP data, inviter output files, Ethereal, and garbage

collection data from the JVM. It takes the available data to generate a PerfColumn for each attribute. The SIP test run data contains the nmon data for all machines, inviter output data from the driver machines, and all the verbose garbage collection logs from the server and load balancer machines which can be used by PET to generate the PerfColumn for the graphs.

From the execution of PET, a set of graphs was obtained from the SIP performance data for each test run. These graphs show the behavior of the system and compare the results with other servers. The following list contains the graphs generated from PET for the SIP project: CPU Utilization (SIP Servers, Load Balancers, and all machines), Garbage Collection Freed Memory, Garbage Collection Size Needed, Memory Available, Memory Swap Space (SIP Servers, and Load Balancers), Total Disk Transfer (SIP Servers, Load balancers, Driver, and all machines), and Network Traffic (SIP Servers and Load Balancers).

Using these graphs, the user can easily identify the behavior of the system through examining the different attributes. It also compares the different servers and load balancers on the same graph for abnormal activities. For a more concrete number for the different attributes describing the system's behavior, the user can look at the calculated values in the PerfColumn files. Using these graphs and PerfColumn, the system behavior can be described thoroughly and performance problems can be identified. With all the tools and environment setup, the next section describes the actual performance tests that were done in order to study the behavior of the system.

3.5 Performance Test Runs

The actual test runs were performed for both the standalone configuration and the simple cluster configuration. For performance modeling purposes, the test runs were done with a single driver, single load balancer, and single SIP server hardware configuration. The results from the test runs were collected for analysis and data reduction.

Due to the nature of the tests, automation scripts were used to simplify the collection and analysis of the data. Each test run took about 15 to 20 minutes to execute. At the beginning of every test, the scripts reboot all the machines used in the test environment to ensure test machines are in a consistent starting state. This is important for studying the performance of the system.

The test run starts with a short warm-up period. The SIP server requires the Java JIT runtime compilation while WAS requires some time to initialize concurrent garbage collection. This period is defined as the time the system needs to perform the first garbage collection that is completed in one thread. After the first garbage collection, if the JVM detects that the garbage collection is too heavy, the JVM sets the garbage collection as a multi-threaded process. It was tested previously that the first single-threaded garbage collection has a significant impact on the system's performance due to the large amount of resources it uses.

Generally, this warm-up test lasts about 5 minutes and the actual test starts shortly after. After the warm-up, the actual test run executes for about 10 minutes, running at the call rate specified. During this time, nmon is executed to capture the system behavior.

When the time is up, the scripts stop the driver and nmon so that the data can be collected for user analysis. All the data gathered from the automation script runs includes the nmon analysis data for all test machines in the environment, WAS garbage collection logs on the server, load generator outputs during warm-up and actual run, and an ethereal capture on one of the test machine.

For each system configuration, four different call rates were used to obtain the system throughput. The first call rate is the one call per second test. This test gives the minimal baseline that the system is working at. The performance is measured without any network traffic contention.

The second call rate is when the system is running at the “ideal condition”. This case is identified to be at the maximum call rate when the server is running at less than 50% CPU utilization. Running the server at above this utilization limit for long period of time can cause the system to be unstable. Figure 12 shows an example of the graph of CPU utilization when the system is at ideal condition.

Figure 12: CPU Server Running at Ideal Condition

The third test is the maximum rate that the system can handle without losing a call during the 10 minutes of testing. This is the case when the system is running at maximum capacity and simulates the call capacity when there is a sudden influx of calls during congested hours. This test ensures the maximum throughput with high reliability during this intense period. The system is not meant to maintain at this call rate for an extended period of time.

The last test run is running the system above maximum capacity. This test is used to determine how the system behaves when it is overloaded and is analyzed for testing

purposes. In this test, it can be seen that lots of retransmission and lost calls are present and the network and server is overloaded.

In a standalone test, the environment is set to send messages from the driver directly to the server. This is the simplest interaction in regards to the client-server system presented in the SIP INVITE scenario. In order to build the performance model, it is necessary to obtain the time spent in the different modules. An Ethereal capture is run on both the driver and server machine at 1 call per second and at the maximum capacity. These Ethereal capture files are then analyzed to obtain the performance numbers for the call flow.

For a clustered environment test, the load balancer is added to the environment between the driver and the server. The call flow for this test is the same as the one used in the previous configuration. For this test, the automation test was used to capture the behavior of the system and the Ethereal capture was done on the driver, load balancer, and server. These Ethereal capture files are used to construct the performance model for the cluster environment.

The data captured from the automation scripts contains many different logs and traces generated from the test runs. From the Ethereal capture files, there is also a large amount of packets that are captured within a few seconds of each test run; thus, there is a lot of information that is needed to be analyzed. The next chapter describes some of the tools and methods used to extract the relevant information to build the performance model and to analyze the results from the test data.

Chapter 4.0: Analytic Model Construction

The benchmark runs in the preceding chapter are useful for gathering data on an existing system; however, the results gathered do not permit the analysts to neither locate performance problems, nor to predict the effects of changes to the system. Rather, they can be used to generate a performance model which can be used to locate bottlenecks and predict the effects of changes.

This chapter describes the construction of a performance model of the SIP stack developed at IBM. In order to do this, the data generated by the various tools used to measure the system must be scaled down to be suitable for a performance model. The first section that follows describes the model itself, derived from the call flow sequence diagrams shown earlier in Figure 9 and 10. Next, the parameters for the model are found primarily by examining the Ethereal packet analyzer data. Finally, the models are calculated, solved and compared against live system data.

4.1 Building the Performance Models

This section describes the construction of the performance models for the standalone environment and the simple one-node cluster environment through a series of steps. A layered-queuing network model is used because the SIP system has a client-server-type architecture and the layered queuing networks technique is particularly well suited to solve this type of model. The SIP models are classified as open systems since

there are essentially an infinite number of users who arrive to the SIP server at random intervals. Although the system is open, a closed model was also used to find the system's theoretical throughput limit.

The first step to constructing the performance model is to list the different components of the SIP system such as the processors, tasks, and the corresponding entries. The standalone environment contains a driver and a server machine, each of which is represented as a processor and task in the LQN model. Since the driver simulates the callers that make SIP calls, this processor represents the users for this system. The driver only simulates the users using the SIP system; thus, the behavior of the driver is not studied in depth. From the hardware configuration described earlier in section 3.2, each of the machines contains four processors. Finally, the WebSphere Application Server installed in the server machines contains ten threads.

With all the components listed, the next step is to identify the interactions in the SIP call flow as described in the Invite scenario. Since all the calls for a SIP invite interaction can be represented in an UML sequence diagram, different techniques can be used to automatically transform the sequence diagrams into an LQN model [Gu05]. For a more generic approach, the SAME technique [Israr05] can be used to examine the communication patterns in the call flow.

Using the SAME technique, the different messages in the call flow represent the traces that are used to generate interaction patterns. Since most of the SIP messages are initiated from the driver user agent and expect a reply, this satisfies the requirements as a synchronous interaction pattern. The following describes the steps used in the construction of the performance models in the standalone and cluster environment.

4.1.1 Standalone Environment Model Construction

From the basic standalone call flow shown previously in Figure 9, it is possible to identify all the messages using the interaction patterns. In the standalone model, there are a total of four synchronous patterns initiated from the user: one “INVITE” message, two “300 Multiple Choice” messages and one “200 OK” message. For the first three messages, the synchronous calls end when the “INVITE” message is sent back to the user. For the “200 OK” message, the driver expects to have another “200 OK” message as a reply.

From the call flow, there are other messages that are not part of the synchronous messages such as the “100 Trying” message and the two “ACK” messages. These can be classified as asynchronous interaction patterns since they are sent from the server back to the driver and a reply is not expected. These four synchronous interaction patterns and the three asynchronous interaction patterns can be transformed into entries to tasks in the performance model. Figure 13 shows part of the sequence diagram from Figure 9 showing which message are part of the synchronous interaction pattern and which message is the asynchronous interaction pattern. This pattern corresponds to the synchronous request on the entry *user to call1* and the asynchronous message from entry *call1* to *port5071*.

Figure 13: Sequence Diagram of First Interaction

Figure 14 shows the LQN model generated for the SIP INVITE scenario based on the First Interaction described in Figure 13.

Figure 14: LQN Model of the First Interaction

The large parallelograms represent tasks. The small parallelograms within the tasks are called entries and serve to differentiate service within a task. The circle in the model represents the processors. The arcs between entries with the solid arrow heads, i.e. from entry *user* to *call1*, are synchronous requests. The arcs with open arrow heads are asynchronous interaction. For example, messages 1 and 3 in Figure 13 are the synchronous call from entry *user* to entry *call1*. The arcs going from task to processors associate the task to its processor and may be omitted on the figure if the tool is the only task running on a processor.

The rest of the call flow shown in the Invite scenario from Figure 9 contains sequences similar to Figure 13. Similarly, these message sequences can be mapped into different model components just as it was done for the first sequence shown in Figure 14. Combining the different components of performance models generated from the different SIP messages completes the whole model of the scenario.

In this model, there is also a need to include the time spend in the network. The network is also represented as a task and a processor module in the performance model. Another synchronous interaction pattern is added from the user to simulate network latency that appears in the system. Based on Ethereal measurements and some calculations, the service times are calculated as shown in a latter section 4.2.4.

Figure 15: SIP Standalone Model

With all the components identified, the full model structure is shown in Figure 15. This model shows the basis of the performance model. The entry *net* is represents the network delay. Entry *call1* represents the initial “Invite” message (message 1 on Figure 9) and the reply of another “Invite” back to the driver (message 3). After entry *call1*, entry *call2* represents the first “300 Multiple Choices” message (message 4) with the “Invite” message reply (message 6). Next, entry *call3* represents the second “300 Multiple Choices” message (message 7) with the reply of the “Invite” message (message

9). Finally, the entry *call4* presents the “200OK” messages to end the call (message 10 and 11).

The remaining three tasks are asynchronous calls from the server back to the driver. The entry *port5071* represents the “100 trying” message (message 2). The entry *port5076* represents the “ACK” message from the first Multiple Choices message (message 5) while the entry *port5074* represents the “ACK” message from the second Multiple Choices message (message 7). This concludes the construction of the standalone performance model. The definition of this standalone model can be found in Appendix B-2.

4.1.2 Cluster Environment Model Construction

The performance model for the cluster environment is more complex as a load balancer is added to the system. The overall call flow is the same as the one for the standalone environment, except all requests and replies are routed through the intermediate load balancer; therefore, the model needs another layer to show this behavior. The initial request messages are sent from the driver to the load balancer, and then the load balancer sends these messages to the server. The reply goes back to the load balancer is then sent back to the driver machine. All the asynchronous messages in the call flow are also sent back through the load balancer from the server. Similarly, the cluster model can be generated using the SAME technique.

The load balancer machine used in this scenario resides on the same LAN segment as the driver and the SIP server. As a result, there is an increase in the amount

of Ethernet traffic as most messages are sent twice. (Please refer back to Figure 7 for the deployment diagram for the cluster environment.) The number of calls that are made to the network task in the model is modified to reflect the additional traffic and is represented as a synchronous interaction pattern from the user to the Ethernet module.

The next set of interactions is the ones made from the user to the load balancer. From the functionality of the load balancer, it either forwards a message from the driver to the server or from the server to the driver. In this case, the load balancer task in the performance model is in a loop and can cause deadlocks trying to handle the two sources of messages. The way to avoid this loop is to separate this task into two pseudo-tasks (i.e. *balancer up* and *balancer down*). Each of these task services one type of messages. These pseudo-tasks are infinite servers with no service times. The messages can pass through these tasks without being blocked. From the pseudo-tasks, they make a synchronous call to another balancer task where the service times are consumed. Using this method, the load balancer's behavior can be modeled.

The task for the SIP server is also present in this cluster model. As configured by the user, there are ten threads set for this task. This task takes care of all the service times for the SIP server. This task then sends messages to the outbound traffic task of the load balancer where this task forwards these messages to the driver.

The User task makes a total of four synchronous calls to the balancer down task of the load balancer. This simulates the messages from the user to the load balancer. This pseudo-task then initiates four synchronous interactions to the real load balancer task where the real service time for the load balancer is located. The pseudo task then

sends four synchronous interactions to the SIP server which represents the messages that are sent from the load balancer to the SIP server.

As shown from the call flow, there are three asynchronous messages that are returned back to the driver through the load balancer. The server task sends three asynchronous interactions to the balancer up pseudo-task and then this pseudo-task sends three synchronous messages to the real load balancer task which contains the service time. This pseudo-task then sends three asynchronous messages back to the driver tasks. Figure 16 shows the basic structure of the cluster model.

Figure 16: SIP Cluster Model

Some of the tasks and entries in this model are the same as the ones in the standalone model. Table 1 displays the model entries and the message sequence it represents from the call flow. The definition of the cluster model can be found in Appendix B-2.

Model Entry	Initial Message	Reply	Interaction Type
Net			Synchronous
Server1	Invite (R1)	Invite (P3)	Synchronous
Server2	300 MC (R4)	Invite (P6)	Synchronous
Server3	300 MC (R7)	Invite (P9)	Synchronous
Server4	200 OK (R10)	200 OK (P11)	Synchronous
C1	Invite (P1)	Invite (R3)	Synchronous
R1	100 Trying (P2)	100 Trying (R2)	Synchronous
C2	300 MC (P4)	Invite (R6)	Synchronous
R2	ACK (P5)	ACK (R5)	Synchronous
C3	300 MC (P7)	Invite (R9)	Synchronous
R3	ACK (P8)	ACK (R8)	Synchronous
C4	200 OK (P10)	200 OK (R11)	Synchronous
Port5071	100 Trying (R2)		Asynchronous
Port5076	ACK (R5)		Asynchronous
Port5074	ACK (R8)		Asynchronous

Table 1: Cluster Model Tasks

This step only describes the structure of the elements in the performance model as it does not include the service times. Finding the service times requires some measurements on the system. From the test runs on the SIP implementation, some measurements data are gathered and the performance values can be extracted. The next section describes the process to obtain the performance parameters to finish the models.

4.2 Model Parameters

The models generated in the preceding section are not useful until performance parameters are entered to describe the system's demand. The process of obtaining performance numbers can be complex. From the test runs, Ethereal captures were done in order to provide data to generate service demands. This section describes the steps taken to obtain the model's parameters. Once these demands are found and entered into the models, the models can be solved.

4.2.1 Ethereal Packets

The PET tool was used to analyze the data that were captured from real system test runs and sort the data into different attributes for the SIP system; however, the Ethereal traces were not analyzed by PET. For building a performance model and finding service demands from the test runs, the Ethereal data has to be analyzed thoroughly to calculate how much time is spent at each stage of a call.

In order to extract the necessary information from the test runs, there is a need to have a method to parse through the Ethereal capture files from the test run to make the analysis simpler. Since the capture file can be translated as plain text, some simple bash scripts were created to parse through the headers of each packet. These scripts capture all the necessary information from the headers and calculate the performance numbers needed to generate the performance model. Figure 17 shows the performance numbers

extraction process and the code of these Ethereal parsing scripts can be found in Appendix C-1 to C3.

Figure 17: Performance Numbers Extraction Process

The process of performance number generation begins with extracting the relevant information needed from the packets in the Ethereal packet captures. In the Ethereal captures, each packet contains a summary line and the packet header field of each protocol layer. The summary line and other important SIP analysis data are gathered into a file on one line for each packet in a spreadsheet format.

Each packet has a summary line containing the time stamp, source IP address, destination IP address, protocol, and the message info. All information is included in the extraction. From looking at the packet header, other important information includes the SIP message ID. This is important since the ID identifies which call the message belongs to. This is similar to the concept of angio-dye that is used for tracking a particular execution [Hrischuk95] except that no changes to the messages are needed as all the information is already present in the protocol and the capture. Multiple calls can occur simultaneously, and the message ID is used to separate them from each other.

After the call flow messages are extracted, they are matched with a template set by the SIP protocol and the scenario of that test that is running. In this case, the template is based on the call flow as shown in Figure 9. This is much simpler than the angio-trace and the SAME technique since there is an expected output of the traces generated. The implementation of this step in the extraction of the information from Ethereal packets is shown in the analyze.sh script in Appendix C-1.

4.2.2 Algorithm for Extracting Service Times

After the information is extracted, each packet from the Ethereal trace can be treated as a trace entry for the test run. The packet trace file contains all the extracted traces generated from the test run. The next step is to separate the messages that belongs to the same call sequence by grouping all the messages containing the same SIP message ID. After separating all the messages, different call flow trace files are generated with the messages of one specific call flow. Each file contains the timestamps of the

messages, and these can be used to extract the time the system required to process the messages. The messages in these call flow traces are also examined to ensure that the call flow was completed without any problems based on the scenario template as shown in call flow diagram in Figure 9.

To compute the service times for the performance model, differences between the time stamps for the packets used in each individual SIP call were computed. For example, the service time for entry *call1* in figure 14 and 15 is the difference between messages 1 and 3. Similarly, the service time for entry *call2* is the difference between messages 4 and 6, entry *call3* is the different between messages 7 and 9, and entry *call4* is the difference between messages 10 and 11.

The driver task accepts asynchronous messages only, so finding the service time may not be precise as it was taken as the time between the arrival of the asynchronous (i.e. message 2 in Figure 13) and the reply message (i.e. message 3 in Figure 13). This method may lead to higher estimates for the service time at the *driver* tasks, but since this task is not important for the model, errors here are acceptable. The time spent at the *driver* task is not important for the analysis of the server, so the time between message 3 and 4 is ignored. The *User* entry is assigned with a service time of zero.

Due to the nature of the a clustered environment and the load balancer, a different set of messages are needed for extracting the Interaction Pattern Times: p1 to r1 + p3 to r3, p2 to r2, p4 to r4 + p6 to r6, p5 to r5, p7 to r 7 + p9 to r9, p8 to r8, and p10 to r10 + p11 to r11. (Please refer to Figure 10 for the call flow messages). Figure 18 show the first set of interaction calls from the main call flow in Figure 10 for the model. These

sets of time patterns are necessary to obtain the service time that was used while the system services the request at the load balancer.

Figure 18: First set of Cluster Interaction

This set of interaction in Figure 18 consists of a synchronous interaction and an asynchronous interaction. The workload in the load balancer consists of the time between the message is received and sent (i.e. P1 to R1). Similar to the standalone call flow, the synchronous interaction (i.e. entry *call1*) consists of message 1 and message 3; thus, the Interaction Pattern Time for this is the time between the timestamp of P1 and R1 plus the time between the timestamp of P3 and R3. Finally, the asynchronous interaction (i.e. entry *reply1*) from message 2 is obtained by using the timestamps of P2 and R2.

Recall in section 4.1.2, the balancer task was split into an up and down part in order to break the cycles in the call graph; therefore, the service times for the synchronous call of the for the balancer task is assigned to entry *c1* in the model shown in Figure 16. Similarly, the service time for the asynchronous interaction is assigned to

entry *r1*. Entries *call1* and *reply1* on tasks *balancer down* and *balancer up* respectively have a service time of zero. The same transformations are done for the remaining messages in the scenario.

Each set of call flow traces can generate a few sets of the Interaction Pattern Times. After extracting these Interaction Pattern Times, the mean can be calculated and be used as the model parameters. These numbers are entered into the LQN model which is then solved by the LQN solver. The solver can then simulate the work load test and predict the behavior of the SIP system. The implementation of the steps from the packet traces to Interaction Pattern Times can be found in `packet.sh` as shown in Appendix C-2 while the load balancer traces have to use `packetSLSP.sh` as shown in Appendix C-3. Using the output from these scripts, the model parameters can be easily calculated and entered into the LQN models.

The three scripts described are used to generate the service time in the performance models in this thesis. The technique described in this section is simpler than other techniques such as SAME is because the SIP protocol already provides a template for the messages from the call flow, the analysis is non-intrusive, and minimal work is needed to gather the data from the test runs. In this thesis, these scripts are written in bash and awk scripting as these tools have the ability to efficiently parse through data from an external file. The following section describes the numbers that were obtained from the execution of these scripts on the capture files obtained from different test runs on the SIP system.

4.2.3 Obtaining Performance Numbers

Using the technique and scripts described above, it is possible to parse the Ethereal capture file obtained from the test runs to obtain the performance numbers to be used in the LQN model. From the model generated, the service times in each call requires a value to be entered. Using the timestamps from the packets in the Ethereal traces, the Ethereal parsing script generates the service time for the messages.

For the standalone system, a short capture test run at one call per second was done to obtain the packets times when there is no contention delay present in the system. This capture was done on the driver and server machines. Each packet is then analyzed and the times between message timestamps are calculated. Table 2 and 3 summarizes the data (in milliseconds) obtained from the scripts analysis on the packets:

Msg.	1 to 2	1 to 3	2 to 3	4 to 5	4 to 6	5 to 6	7 to 8	7 to 9	8 to 9	10 to 11
P. 1	0.71	1.723	1.013	0.427	0.631	0.204	0.325	0.424	0.099	0.558
P. 2	0.539	1.325	0.786	0.385	0.587	0.202	0.344	0.482	0.138	0.524
P. 3	0.517	1.326	0.809	0.394	0.521	0.127	0.311	0.391	0.08	0.545
Avg.	0.589	1.458	0.869	0.402	0.580	0.178	0.327	0.432	0.106	0.542
Std. Dev.	0.106	0.229	0.125	0.022	0.055	0.044	0.017	0.046	0.030	0.017

Table 2: Standalone Message Time from the Driver Capture file in ms

Msg.	1 to 2	1 to 3	2 to 3	4 to 5	4 to 6	5 to 6	7 to 8	7 to 9	8 to 9	10 to 11
P. 1	0.61	1.614	1.004	0.326	0.525	0.199	0.228	0.319	0.091	0.458
P. 2	0.436	1.218	0.782	0.284	0.482	0.198	0.245	0.377	0.132	0.424
P. 3	0.404	1.219	0.815	0.296	0.417	0.121	0.213	0.287	0.074	0.044
P. 4	0.434	1.329	0.895	0.33	0.501	0.171	0.262	0.359	0.097	0.406
Avg.	0.471	1.345	0.874	0.309	0.481	0.172	0.237	0.336	0.099	0.433
Std. Dev.	0.094	0.187	0.099	0.023	0.046	0.037	0.021	0.040	0.024	0.023

Table 3: Standalone Message Time from the Server Capture File in ms

Due to the network delay, it is necessary to perform packet captures on both the driver and server machine to obtain the best estimated time. From the data captured, it can be seen that the service times differ depending on where the messages are being handled. The network service time calculation is discussed in the next section.

With these numbers, it is possible to fill in the service times in the performance models. For synchronous Interaction Patterns, it is necessary to look at the time on the server machine as that is where the service took place. Table 4 shows all the numbers that are taken from the measurement data that are used to fill in the service time in the performance model:

Model Entry	Service Time	Message Number	Ethereal File
Call1	0.001345	1 to 3	Server
Call2	0.000481	4 to 6	Server
Call3	0.000336	7 to 9	Server
Call4	0.000433	10 to 11	Server
Port5071	0.000869	2 to 3	Driver
Port5076	0.000178	5 to 6	Driver
Port5074	0.000106	8 to 9	Driver

Table 4: Service Time for Standalone Model

These service times are entered into the standalone model as shown in the Appendix B-1. After calculating the network load shown in the next section, the model is complete and ready to be solved. Using this technique, the service times for the cluster environment can be extracted.

For the cluster environment, it is necessary to take into account of the load balancer. That means there is an extra Ethereal capture file needed for the analysis. Similar tests were run on the cluster environment and capture files were analyzed using the scripts. Table 5 and 6 shows the numbers from the driver and the SIP server that came from the analysis using the scripts.

Msg.	1 to 2	1 to 3	2 to 3	4 to 5	4 to 6	5 to 6	7 to 8	7 to 9	8 to 9	10 to 11
P. 1	1.526	2.233	0.707	0.813	0.96	0.147	0.675	0.798	0.123	0.763
P. 2	1.428	1.863	0.435	0.766	0.915	0.149	0.66	0.799	0.139	0.713
P. 3	1.202	1.602	0.4	0.766	0.963	0.197	0.694	0.84	0.146	0.712
Avg.	1.385	1.899	0.514	0.782	0.946	0.164	0.676	0.812	0.136	0.729
Std. Dev.	0.166	0.317	0.168	0.027	0.027	0.028	0.017	0.024	0.012	0.029

Table 5: Cluster Message Time from the Driver Capture File in ms

Msg.	1 to 2	1 to 3	2 to 3	4 to 5	4 to 6	5 to 6	7 to 8	7 to 9	8 to 9	10 to 11
P. 1	0.763	1.483	0.72	0.412	0.552	0.14	0.31	0.412	0.102	0.438
P. 2	0.717	1.166	0.449	0.398	0.525	0.127	0.306	0.418	0.112	0.388
P. 3	0.589	0.988	0.399	0.394	0.58	0.186	0.313	0.454	0.141	0.381
Avg.	0.67	1.2112	0.523	0.401	0.552	0.151	0.31	0.428	0.118	0.402
Std. Dev.	0.09	0.251	0.173	0.009	0.028	0.031	0.004	0.023	0.02	0.031

Table 6: Cluster Message Time from the Server Capture File in ms

For the load balancer, the capture extraction procedure is different since the call flow contains double the amount of messages. Since the load balancer acts as a forwarding service in a clustered environment, the messages involved can be inbound and outbound. Using the algorithm described in Section 4.2.2, the different message times needed for the load balancer are shown in Table 7.

Msg.	1 to 3	2	4 to 6	5	7 to 9	8	10 to 11
P. 1	0.553	0.179	0.201	0.097	0.176	0.092	0.141
P. 2	0.497	0.184	0.181	0.089	0.174	0.084	0.14
P. 3	0.422	0.168	0.174	0.092	0.177	0.096	0.147
Avg.	0.491	0.177	0.185	0.093	0.176	0.091	0.143
Std. Dev.	0.066	0.008	0.014	0.004	0.002	0.006	0.004

Table 7: Cluster Message Time from the Load Balancer Capture File in ms

From the three sets of message times, it is possible to fill in the performance numbers into the cluster model. Table 8 displays the service number that is put into the cluster performance model.

Model Task	Service Time	Message Number	Ethereal File
Server1	0.00121	1 to 3	Server
Server2	0.000552	4 to 6	Server
Server3	0.000428	7 to 9	Server
Server4	0.000402	10 to 11	Server
Port5071	0.000514	2 to 3	Driver
Port5076	0.000164	5 to 6	Driver
Port5074	0.000143	8 to 9	Driver
C1	0.000491	P1 to R1 + P3 to R3	Balancer
C2	0.000927	P4 to R4 + P6 to R6	Balancer
C3	0.000176	P7 to R7 + P9 to R9	Balancer
C4	0.000143	P10 to R10 + P11 to R11	Balancer
R1	0.000177	P2 to R2	Balancer
R2	0.000185	P5 to R5	Balancer
R3	0.0000907	P8 to R8	balancer

Table 8: Service Time for Cluster Model

These service times are entered into the cluster model as shown in the Appendix B-2. It can be seen that the service time on the server and driver files are similar to the ones from the standalone environment. The presence of the load balancer does not affect the service time of the server, but it impacts the overall system performance. Further analysis on the cluster environment is discussed in latter sections.

4.2.4 Calculating Network Load

The last parameter required for the performance model is the service time needed for the network entry. In all systems, the network can play a big factor in the performance of the system. Even though this environment uses a gigabit network switch, it is necessary to calculate how much work and time is needed for the messages to process from the call flow.

The calculation of the network has been modeled by a group in University of Washington [Lazowska84]. From this research, the researchers have tried to model the “response time” of messages going from a source node to a destination node. This is done using an opened and closed queuing model with M number of nodes from the source to the destination. The users, represented by messages, enter the system from the source, travel through the nodes, and arrive at their destination. The testing has been done over an isolated local area network where measurements can be made and a few formulas were obtained to calculate the network service time.

The implementation of the Ethernet is very complex, but the simple underlying policy that it is based on can be incorporated it into a queuing network model

[Lazowska84]. This model is able to predict the efficiency of Ethernet as a function of instantaneous load and can be used for the service time of the network module in a system-level model. From this research, a few formulas have been published to calculate the network latency that can be used as service times for the performance model. These formulas look at slots S where it represents the time duration of a single round-trip propagation in the network based on n stations in the channel wishing to transmit data.

If n stations are to transmit data independently, the probability that a station can transmit during the time slot S is $1/n$. With this, the probability A that any of the stations successfully acquires the channel for transmission can be calculated with Equation 1:

$$A = \left[1 - \frac{1}{n} \right]^{n-1} \quad (1)$$

This probability is then used to calculate the average number of slots devoted to contention (or collision) C before a station obtains the channel. This can be calculated with Equation 2:

$$C = \sum_{i=1}^{\infty} i A (1-A)^i = \frac{1-A}{A} \quad (2)$$

The efficiency calculation can be done by using the length of a transmission interval divided by the sum of the length of a transmission interval and the length of a contention interval. The length of a transmission interval can be calculated as the average packet length P in bits divided by the network bandwidth B in bits per second. The length of a contention interval can be calculated as the number of slots devoted to contention C multiplied by the slot duration S . The parameter S is stated from the physical configuration relating to the length of the network. Typically, the value for S is 0.0000512 seconds. Equation 3 shows the efficiency calculations using the different parameters as described:

$$E(n) = \frac{P/B}{P/B + C \times S} \quad (3)$$

To generate the network service time for the SIP performance models, it is only necessary to calculate the length of the transmission interval plus the length of the contention interval. This gives the service time on the network due to contention and transmission time. The calculation for calculating the service time for the network task for the standalone and cluster environment was done using the formulas discussed earlier.

For the standalone environment, there are 2 machines present. That gives n a value of 2 for the number of stations. Using Equation 1, A gives a value of 0.5. Using this value and Equation 2, it yields C the value of 1. With this value, Equation 3 can be used to obtain the service time for the network. To find the value of P , the mean is taken

from all the packet sizes from the Ethereal traces. Table 9 gives a summary of the packet size as observed from the Ethereal packet capture.

Packet	1	2	3	4	5	6	7	8	9	10	11	Avg.
Size(bytes)	378	296	490	434	368	490	434	368	490	420	389	414.27

Table 9: Packet Sizes from the Standalone Environment Ethereal Traces

The average packet size in the capture is about 414.27 bytes or 3314 bits, giving the value of P . The network bandwidth B is 1,000,000,000 bits per second because the blade chassis are connected through a gigabit switch. With the four parameters P , B , C , and S , the service time can be calculated by using " $P/B + C \times S$ " from Equation 3 which gives 0.0000545 seconds. This service time is entered into the standalone performance model for the network entry and the model is now complete.

Similarly, this calculation is done for the cluster environment. Since the load balancer is present in the network, it is necessary to define n as 3 as there is the extra station present in the network. Using Equation 1, $A = 0.444$. Similarly, Equation 2 provides $C = 1.25$ for the number of slots devoted to contention. With Equation 3, it is necessary to look at the Ethereal traces to calculate the average packet size. Since there is a load balancer, the number of packets present is doubled from the standalone environment as shown in Table 10.

Packet	1	2	3	4	5	6	7	8	9	10	11	
Size(bytes)	378	459	399	296	595	718	662	580	370	493	594	
Packet	12	13	14	15	16	17	18	19	20	21	22	Avg.
Size(bytes)	717	661	579	369	492	594	717	647	565	494	391	535

Table 10: Packet Sizes from the Cluster Environment Ethereal Traces

The table shows that the average size of a packet is 535 bytes or 4280 bits represented by the parameter variable, P . Similarly, the network bandwidth B is still 1,000,000,000 bits per second. Using the four parameters, the service time is calculated to be 0.00006828 seconds. This value is entered into the cluster model as the network entry service time.

Network parameter calculations can be validated through the measurement data that was analyzed from the previous section. Running a test with one call per second, packet captures were made on the same call but on both the driver and server. The service time calculated with the varying timestamps yield varying results which is consistent with network delay. In both standalone and the cluster environment, the difference in delay time is roughly twice the amount. This accounts for the message going to from the driver to the server and from the server back to the driver. This shows that the real system measurement data matches with the theoretical calculation for the network delay.

4.3 Solving the Performance Models

From the previous sections, the performance numbers are generated and entered into the LQN models and then solved using the LQN solver. The output data from the solver is used for data analysis. Please refer to Appendix B-1 for the complete model of the standalone environment and Appendix B-2 for the complete model of the cluster environment.

First, the model of the standalone environment is solved and analyzed. In order to obtain the maximum theoretical throughput, the closed model is used with the maximum number of users the model can support, in this case, two was the maximum number that can be put into the model. This gives the maximum throughput of the system as the driver is sending as many requests as possible. After solving the performance model, the graphical display of the results shown in Figure 19 was obtained and displays the total throughput and utilization of each module.

Figure 19: LQN Solver Output from the Original Standalone Model

From the output, the throughput is predicted to be about 604 calls per second (cps) with the bottleneck at the *driverport* task where this spot is heavily utilized. This can be seen in the output file put in Appendix D-1. Similarly, the solver was used to run the cluster performance model using 20 users as shown in Figure 20.

Figure 20: LQN Solver Output from the Original Cluster Model

From the output as shown in Appendix D-2e, it predicted that the maximum throughput on this system is about 665 calls per second and the bottleneck is concentrated in the network. Theoretically, the models predicted how the system should behave and what the maximum throughput should be coming out of the system. The following section summarizes the results that were observed from the real system performance runs.

4.4 Measured Performance Data

From the system test runs performed in the lab, results were obtained for both the standalone test and the cluster test at different call rates. These are the call rates that were used on the actual system for testing based on different scenario criteria. These tests are based on a 10 minute load test and a standard system warm-up before the start of the load test. Many test runs were performed for the different scenarios such as one call per second, ideal condition where CPU utilization never goes above 50%, maximum throughput with the maximum call rate achieved without any packet loss, and when the system is saturated.

These tests were performed on the system with a clean reboot to minimize any unwanted residual data in the memory as described in section 3.5. The maximum call rate for each condition was obtained through testing and analyzed at different call rates. Only the maximum call rate that satisfied the test conditions was recorded and analyzed with the performance models. With each performance test run, the PET tool was run with the captured data to generate suitable graphs to be used for analysis.

Through testing on the standalone environment, the following call rates and data capture file was obtained. It also shows other information such as the CPU utilization on the machines that was analyzed through the PET analysis.

Condition	CPS	Server Avg.	Server Max	Server SD	Driver Avg.	Driver Max	Driver SD
1cps	1	0.774	15.5	1.72	0.605	4	0.426
1cps	1	0.791	11.9	1.69	0.545	3.3	0.374
Optimal TP	210	17.4	44.5	4.18	1.27	4.5	0.552
Optimal TP	210	17.4	43.3	4.23	1.26	23.6	0.597
Max. TP	350	31.5	70.1	8.2	1.73	5.5	0.627
Max. TP	350	31.2	75.2	8.5	1.85	20.2	1.3
Network Storm	600	40.4	77.2	7.43	3.7	14	1.15
Network Storm	600	40.0	77	7.59	3.56	11.1	0.945

Table 11: Call Rate and CPU Utilization for Standalone Real System Measurements

Table 11 shows the call rate and the CPU utilization of the system from actual measurements. Similarly, the measurements and analysis was done on the cluster environment. Table 12 summarizes the call rates and the CPU utilization on the machines that were involved in the cluster environment.

Condition	CPS	Server Avg.	Server Max	Server SD	LB Avg.	LB Max	LB SD	Driver Avg.	Driver Max	Driver SD
1 cps	1	0.814	13.7	1.51	1.51	9.9	2.13	0.577	3.4	0.374
1 cps	1	0.751	13	1.49	1.75	17.8	2.72	0.62	3.2	0.424
Optimal TP	200	18.6	49.8	7.46	7.46	21.1	2.94	1.36	5.3	0.601
Optimal TP	200	19.1	52.7	7.48	7.48	17	2.6	1.32	5.7	0.558
Max. TP	290	28.3	63.7	10	10	21.3	2.97	1.66	6.1	0.657
Max. TP	290	27.7	73	10.1	10.1	21.9	2.92	1.6	0.657	0.657
Network Storm	500	37.73	66.5	21.6	21.59	34.9	3.46	3.459	12.4	1.08
Network Storm	500	38.35	74.7	20.1	20.12	35.3	3.1	3.098	12.5	0.945

Table 12: Call Rate and CPU Utilization for Cluster Real System Measurements

The observed real system data can now be used for comparison against performance numbers generated from the performance models. The goal is to have the predicted throughput from the performance model to match with the real system data. The following section explains the details on the real system analysis measurements with the performance model generated in the previous section.

4.5 Model Analysis

With the prediction from the performance model generated and the real system performance data, it is possible to compare and validate them to see if the numbers match up. Table 13 summarizes the two sets of data as discussed from the previous sections.

Environment	Real System Throughput	Model Predicted Throughput	Predicted Bottleneck
Standalone	350 cps	604 cps	Driverport
Cluster	290 cps	665 cps	Network

Table 13: Comparison of Real System Data with Model Predicted Data

For the standalone system, the predicted output is much higher than the actual real system data observed. This proposed that there are some discrepancies with the performance model. The bottleneck proposed from the performance model in Figure 19 states that the driver task is being driven to a very high utilization. The processors of the model are approaching a total of 70% in CPU utilization. Looking at the CPU utilization from the real system data, it shows that the utilization on the driver machine averages to about 0.5% to 0.6%. The real system data shows that the driver module should not be the bottleneck of the system. Table 14 shows the comparison for the different CPU utilization of the standalone system.

Data	Driver	SIP Servers
Performance Model	0.697 -> 17.4%	1.569 -> 39.2%
Real System	1.8%	31%

Table 14: CPU Utilization Comparison for the Standalone Environment

Another area to look at is the CPU utilization on the SIP servers. During the setup, it is known that 2 processors with hyper-threading were available on each of the machine. This shows that the operating system sees the machine in having 4 processors.

In the performance model, the predicted SIP Server is 157% over 4 processors or about 40% utilization on each processor. In the real data measurements, it shows that the CPU utilization for the SIP servers is about 31% on each of the processor. This shows that the CPU is more heavily utilized than the predicted model. From the analysis of the standalone model, it shows that the maximum throughput is too high but the SIP server utilization is too low with the driver module being the bottleneck of the system.

For the cluster system, the predicted output is also much higher than the actual real system data observed. From the model in Figure 20, it shows that the network is the bottleneck since the network module is heavily utilized. The load of the network is much heavier due to the extra messages that are being sent within the cluster environment. This shows that the system has reached the maximum capacity with the network being completely saturated.

Another area to look into is the CPU utilization measurements on the different machines. Table 15 summarizes the numbers from the model and real system data.

Data	Driver	Load Balancer	SIP Server
Performance Model	0.5417 -> 13.5%	0.6621 -> 16.6%	1.725 -> 43.1%
Real System	1.6	10	28

Table 15: CPU Utilization Comparison for the Cluster Environment

As shown in Table 15, the predicted CPU utilization on all three modules from the performance model is much higher than the expected values. Since the maximum throughput of the system is also much higher than expected, there must be another

bottleneck that saturates the system before the network saturates as predicted, possibly other areas in the system where it is causing some problems.

The analysis in this section shows that the performance models predictions are much higher than the actual real system data in terms of maximum throughput. The next step is to discover the reason for the discrepancy between the performance models and the real system data. The following chapter lists some possibilities that might affect the results generated for performance models in this thesis and the models are calibrated.

Chapter 5.0: Model Calibration and System Predictions

As shown in the model analysis in Chapter 4, the SIP models generated from the captured data do not match with the real system captured data. In order to obtain similar results, some calibration and adjustments are required on the performance models. This section describes some of the changes made to the model so that the overall throughput and system behavior predicted is similar to real system data. This section also gives some details on capacity planning in regards to the SIP cluster configurations and other changes that could improve the overall performance.

5.1 Factors that Might Affect Results

There are different factors that might affect the test measurements in the actual system that can cause extra overhead to the system and increase unwanted workload in the system. The following factors have to be taken into consideration for any discrepancies in the model as the results are being analyzed.

The first factor that has to be taken into consideration is that the network in the real system is not isolated. There are other packets that may travel through the same network to the various nodes that reside in it. For example from looking at the Ethereal traces without any filtering, there would be broadcast traffic, NetBios Name Service (NBNS) traffic, and DNS traffic within the system to name a few. These unwanted network traffic are unnoticeable during the test as there is only an average of one or two

of these packets every second and they are very small. Even though each packet itself is small, it is possible that these can cause unwanted workload in the system when there are many of these packets are accumulated.

Another factor that should be taken to consideration is that when running a system test at a higher load, there are significantly more SSH packets present in the system. The extra network overhead causes more traffic within the network while creating more contention between the packets that are traveling in the same network. Due to the nature of the test setup, this cannot be avoided and has to be taken into consideration on the results obtained.

The collected data from the test runs that was used to generate the performance numbers are based on a small sample of data during the packet capture. There is a margin of error in the calculation for the service times as shown in the standard deviation from the result tables. For the analysis, the comparison was made to see if the performance model data is similar to real system data. With this margin of error, comparison may not be very accurate.

Even from considering the above possibilities as factors that may affect the test results, they still do not reveal why the system causes such a big discrepancy between the performance models and the real system data observed. It seems that some key changes are needed to be made in the model. In order to do this, some model calibration needs to be done on the performance models in order to yield results that may match the real system data. The next section discusses the process of model calibration.

5.2 Objective of Model Calibration

Using the models generated from the prototype, the predicted behavior was compared with PET's real system capture analysis. The analysis shows that the behavior predicted by the model does not agree with the real system data. This happens when the real system is not behaving as expected or there are some attributes that cannot be easily and accurately measured. There could also be some problems with the system configuration that causes the system not to run at its fullest potential. Through calibration, it is possible to find where the problems lie and then modify the models to match the two sets of data.

Model calibration is a common practice since all attributes of a system are hard to obtain. It is understood that modeling is only an approximation and the only correct model is the real system itself. Formally, the definition of model calibration means to use a known set of methods to force the results from the generated model to match with the actual results. Some sources of potential error that allows users to do this include inaccurate measurements, non-exponential service times, different scheduling schemes, and demands that were not measured by the monitor. These can cause the model to produce very inaccurate results.

In model calibration, there are different ways to change the output or input parameters such as by taking an absolute difference or percentage difference for example. Taking an absolute difference means shifting the parameters by a set amount needed. This kind of calibration is applied to all output parameters to match the real system data.

There are many different techniques that are used to calibrate the model [Manasce94]. Different types of models need different calibrations to be done. It is also possible that calibration methods cannot successfully match the two sets of data. Generally, there are a few rules that are given for model calibration. First, changing the demands on the bottleneck devices is usually more accurate than changing the demand on other components since the bottleneck is the most sensitive component of the system. Also, changing the outputs of the model is usually more effective than changing the input parameters. The next section shows some techniques that were used to calibrate the SIP performance models to force the predicted data to mirror real system data.

5.3 *Calibrating the Models*

From the environment configuration described in section 3.3, the models for standalone and cluster environments were built according to the hardware and server setup specified. After solving the models, the maximum call rates from both performance models were much higher than the actual measurements from the real system. After checking the model, it was believed that all the numbers were correct according to the specifications; therefore, there must be another bottleneck in the system that is causing this discrepancy. The first step is to find a place where it could most likely be a problem. In order to lower the system throughput, the service time demand in the bottlenecks needs to be higher, or the amount of resources needs to be lowered. These calibrations are performed on the standalone and cluster environments as shown in the following sections.

5.3.1 Standalone Environment Calibration

From the analysis, it shows that the model predicted throughput is too high while the CPU utilization on the SIP server is much lower than expected. With such a big difference in the observed behavior, increasing the service time only ends up in a value that is much larger than the measured values from the actual system. The easiest way is to decrease the resources to eliminate the multi-processor for the server machine. From observing and studying the SIP environment, it is possible that the Java Virtual machine only used one processor on the node instead all four of the processors. Using one processor calibrated into the model while keeping the rest of the model properties from the original model, the bottleneck moved to the SIP server as observed in the real system. Figure 21 shows the standalone model after the number of processors has been decreased to one. The actual calibrated standalone model implementation can be found in Appendix B-3.

Figure 21: Calibrated Standalone Model Output

This calibration causes the SIP server to be fully utilized; thus, making the SIP server to be the bottleneck of the system. The maximum throughput decreased to about 370 calls per second which is similar to the real system data. After this calibration, the predicted behavior from the standalone model and the actual system data are matched and the model is validated and complete. Next section describes the calibration done for the cluster environment.

5.3.2 Cluster Environment Calibration

From the standalone calibration, similar calibration was performed with the processor in the cluster environment model. First, the number of threads in the load balancer is reduced to 1 thread instead of 10 and the number of processors has been reduced. Figure 22 shows the result of the performance model using these similar calibrations.

Figure 22: Calibrated Cluster Model Output

This calibrated cluster model implementation can be found in Appendix B-4.

From the model output, it also shows that the new bottleneck is the SIP server as observed from the new model. The throughput is still a higher than expected but the system utilization is pretty close to the real system data. There could still be a bottleneck

somewhere such as the Ethernet that is not observed from the analysis. Next section gives a more in depth comparison between the calibrated model and the real system data.

5.4 Calibrated Model Analysis

The calibrated models show that both of the model's system performance is bound by a different bottleneck and the overall throughput is reduced. Similarly with the previous chapter, a comparison is done with the predicted data and the real system data.

Table 16 summarizes the overall throughput of the system.

Environment	Real System Throughput	Calibrated Model Predicted Throughput	Predicted Bottleneck
Standalone	350 cps	371 cps	SIP Server
Cluster	290 cps	375 cps	SIP Server

Table 16: Comparison of Real System Data with Calibrated Model Predicted Data

The comparison above shows that the real system throughput and the predicted throughput from the calibrated model are very similar with the standalone model (i.e. within 10% difference from the actual value). For the cluster environment, it shows that there is still a big difference between the real system throughput and the predicted throughput. Further analysis is done in the next section on the cluster environment to determine what may cause such a large discrepancy.

The models are studied further by looking at the CPU utilization predicted from the calibrated model and compared with the real system data. This is shown in Table 17.

Data	Driver	SIP Servers
Calibrated Performance Model	0.4278 -> 10.7%	0.9628 -> 24%
Real System	1.8%	31%

Table 17: CPU Utilization Comparison for the Standalone Environment

From the table above, it shows that the predicted CPU utilization is quite similar with the real system data for the SIP server. Even though the utilizations are not actually close in general, the overall system throughput obtained is quite close to the real system value. The reason due to the difference is that the real system data contains the work that is being done by the system other than the work from the SIP test. The model does not account other resources used by the system but this is reflected in the real system measurements. Also, the utilization of the driver is not important as this is a study on the SIP server. This analysis proves that the procedure of cutting down the number of processors from 4 to 1 is a valid change for the model.

Table 18 shows the CPU utilization from the calibrated model prediction and the real system data for the cluster environment.

Data	Driver	Load Balancer	SIP Server
Calibrated Performance Model	0.306 -> 7.65%	0.374 -> 9.35%	0.9743 -> 24.4%
Real System	1.6	10	28

Table 18: CPU Utilization Comparison for the Cluster Environment

The table shows that the CPU utilization from the prediction of the calibrated model is quite similar to the real system data. It shows that the model is pretty close to the expected behavior in terms of the resource usage. The changes show that the system is being more fully utilized at the SIP server from the original model. The utilization is still high but it is due to the extra CPU resource usage outside the SIP server. In terms of system throughput, this model should give a more precise number; thus, there seems to be another bottleneck still present in the system.

From analyzing the system, there seems to be no other parameters that can be easily changed without making a drastic shift in values to the parameters. Other than the system, the only other area left to look at is the network and network modules. There might be a bottleneck there that causes the system not to be fully utilized. The next section suggests an idea of why the network might be a bottleneck in the system.

5.5 Maximum Gigabit Network Capacity

After calibrating the models, there seems to be another factor that contributes as the bottleneck for the cluster environment. Since the system CPU resources are already calibrated as much as possible, the only other area to look at is the network. As stated in

section 2.2.3, Ethernet is a difficult area to model due to the CSMA/CD nature. Under a heavy load, the simulation turns to be very complex and very hard to predict the behavior. Due to the complexity, a few different cases of the network maximum performance have been studied.

From the work of different researchers, they came to a conclusion that the actual maximum throughput is much lower than the theoretical value for the gigabit network capability [Shaikh01] due to the nature of communication protocols. This is especially true for the SIP system since SIP request messages and their replies are time-delay sensitive. The SIP system expects a reply to return after a certain amount of time. If the reply fails to return within a set time frame, the sender generates a retransmission or a packet loss message.

In SIP testing, a test run was performed while the load driver sends INVITE messages at a rate much higher than the maximum capacity of the system. During this test, it can be seen that the system is fully saturated and most of the INVITE messages were not successful. The reason is due to the amount of traffic and collision in the network from nodes trying to access the network at the same time. Since the messages did not arrive within the allowable time frame for package acknowledgement, an extra load of messages from retransmissions and packet loss are returned and sent within the network. This causes more overhead to the overall network and results in complete network saturation. With the amount of undesirable messages, the system is in the state of a “network storm” where many messages are transmitted but most of them do not end up in a successful call flow.

Using the SIP protocol and other transport protocols, there is a time requirement on the responsiveness of the network to avoid retransmission and packet loss. In most cases, Gigabit Ethernet has the capacity to only handle up to a certain amount of data transmission that is much less than the line capability. An example is found in the Cisco 7500 Gigabit Ethernet Interface Processor (GEIP) white paper where it tested the performance at the Layer 3 packet switching [Cisco98]. This is important as the SIP environment setup uses Cisco's network hardware with the blade servers. The Cisco's study used a Gigabit network that can have the maximum line throughput of 1000 Mbps. Different tests were performed using different packet sizes on the Ethernet to measure the packet switching performance. The results from these tests were very similar to the SIP testing as the maximum throughput on this Ethernet switch was 40% of the maximum theoretical rate. From this experiment, it shows that the GEIP was only able to handle the packet switching successfully at the maximum throughput of 40% of the network capability.

A similar Ethernet test was performed by Farrell and Ong [Farrell00] at Kent State University where they tested the TCP/IP characteristics on a Gigabit Ethernet network under a cluster computing environment. The tests were done in a Linux environment with a software program called NetPipe to test the communication performance. This program transmits different block sizes of requests from small to large until the transmission time exceeds one second. Other than the hardware and software setup, the Linux kernel was tuned and tested to provide the best results for TCP/IP.

The purpose of this performance test is to show how TCP/IP behaves on the Gigabit network. The different test parameters are based on the packet block size and the

tuning of the kernel. They also conducted the same performance tests on different drivers and processors. From the observations in these tests, the maximum throughput of the system seems to be under 400 Mbps per switch. This is very similar to the other test that was performed on the Cisco equipment.

There is a limit to the capability of the Gigabit Ethernet for the use of communication. There are researches that are being done to improve the overall throughput of the network. An example is the GigaE PM [Sumimoto98] setup that has proposed by Sumimoto et al. It takes advantage of virtual networks and parallel data processing in the Network Interface Card hardware. This network facility was able to reach a higher overall communication throughput but was still not able to fully utilize the network while giving reliable communication results.

From these cases, it can be seen that the maximum throughput at the TCP/IP protocol level is about 400 Mbps or 40% of the maximum capacity in the Gigabit network. The SIP cluster performance model shows that the utilization of the cluster environment is about 56%. This could be the reason that the system was not able to handle the network throughput due to the capacity and timing constraint at the Transport Layer. In order to demonstrate the actual system model behavior, the throughput needs to be lowered as the network sets another constraint. With the cluster environment, an open model is generated with the real-system observed throughput to describe the behavior of the system. Figure 23 shows this open model using a call rate of 290 cps. The actual source of this open model can also be found in Appendix B-5.

Figure 23: Open Model of the Calibrated Cluster Model at Maximum Throughput

This open model uses a throughput of 290 cps which is the observed maximum from the real system. With this new constraint, it shows that the network utilization is about 44% which is very close to 40% from the previous studies. In this situation, Table 19 gives a summary of the CPU Utilization.

Data	Driver	Load Balancer	SIP Server
Calibrated Performance Model	0.236 -> 5.9%	0.289 -> 7.2%	0.752 -> 18.8%
Real System	1.6	10	28

Table 19: CPU Utilization Comparison for the Cluster Environment

The CPU utilizations in the table above show that the CPU utilization is a little lower than the real system measurements but the network utilization is close to the constraint under the throughput from the real system measurements. The experiment shows that the model was able to predict the behavior of the SIP system in the standalone and cluster environment as the throughput and utilization are quite similar between both the model data and the real system data. From observing the standalone environment, the maximum Ethernet limit was not present as the CPU saturates on the standalone SIP server with the network utilization at only at 22%.

5.6 Capacity Planning and System Improvements

Using the calibrated models, it is possible to perform capacity planning for the SIP system. Capacity planning is the process of determining and allocating the amount of hardware and software for a system to handle a required projected traffic load. With IBM's WAS cluster environment setup, it is very easy to add resources to the system as WAS is designed to allow easy addition of server nodes to the cluster. This is very useful for capacity planning. In this thesis, some capacity work has been done in regards to the SIP cluster environment as modeled from the previous chapters. Testing was also done on a configuration that was able to fully utilize the SIP system.

Adding more machines in a clustered environment gives additional processor to handle the work required for the system as a whole. In a cluster environment, it is very easy to add an extra SIP server or an extra load balancer to share the workload. At this point, the total throughput increases due to the sharing of resources. In planning how much work the system can handle, some test runs were done based on adding more SIP servers and load balancers.

In this thesis, the same Invite tests were done with different cluster configurations. Table 20 gives the maximum throughput of the system using different numbers of load balancers and SIP servers. A separate driver machine was used for each load balancer present in the system. This shows how the system behaves when more machines are put into the cluster.

	1 Load Balancer	2 Load balancers
1 SIP Server	290 cps	130 cps (260 cps total)
2 SIP Servers	570 cps	220 cps (440 cps total)
3 SIP Servers	850 cps	335 cps (670 cps total)

Table 20: Maximum Throughput of Different Cluster Configurations

The table shows that by adding more SIP servers, the system was able to handle more calls. As an additional SIP server was added, the total throughput of the system increased by around 290 cps. It can be seen that the all the SIP servers' utilization in the different setups are the same among each server and for the case when there is only 1 SIP server – maximum of 28% CPU utilization on the server up to three different server machines included in the same cluster. The only increase in the utilization is at the load balancer (24%). This proves that by adding more processors at the SIP servers, the overall system was able to handle more calls. Figure 24 shows the CPU utilizations for the cluster with 1 load balancer and 3 SIP servers.

Figure 24: CPU Utilization for Cluster with 1 Load Balancer and 3 SIP Servers

When an extra load balancer is added to the system, the overall throughput significantly dropped due to the extra overhead in the SIP server to redirect all the packets back to the sending load balancer. It shows that adding the extra load balancer causes extra overhead to the system and it should only be added when the load balancer becomes the bottleneck of the system. Since the bottleneck is the SIP server from the analysis, adding more load balancers does not help to improve the overall system performance.

One of the main purposes of performance engineering is to be able to obtain the maximum throughput with the resources given. This requires changing the configuration of the system to utilize all available resources. The experiment in this thesis shows that the main bottlenecks of the cluster system are the network and the SIP servers. In order to fully utilize the SIP servers, there needs to be a way to overcome the network bottleneck to the server.

WAS is the application server on the SIP machine where it deploys the actual SIP server. WAS also has the capability to deploy multiple servers on the same node so that the resources can be fully utilized. The settings of the servers can be set to use different ports and network interfaces to avoid the network collision and delays which causes the network module to overload. Also, high availability option can be implemented to ensure there is a less chance of failure within the system.

In the final test case, a test was performed with three SIP server machines and two load balancer machines. Each SIP server machine contains two SIP servers and each load balancer machine contains two load balancer servers. This gives a total of four load balancer servers and six SIP servers. The load balancers distribute the workload evenly to the six different SIP servers. With this new configuration, the same set of test runs that were run as previously described in the preceding chapters was performed.

From the test runs, the system showed a maximum throughput of 245 cps per driver or a total of 980 cps. The extra server on the machine causes the processors to work harder and thus the processors are fully utilized. Figure 25 shows the CPU utilization of the machines that are involved in this configuration.

Figure 25: CPU Utilization of the Fully Utilized SIP Cluster System

The graph in Figure 25 shows that the CPU utilization of the SIP servers averages at around 64% while the load balancers' utilization averages at around 32%. For the SIP servers, some of the spikes were able to reach to almost 100%. Using this configuration, it can show that the system is fully utilized when there are 2 servers on each of the machines in the cluster.

With this unique configuration, an LQN model was generated based on this configuration with 4 load balancer servers and 6 SIP stack servers. The model structure and service times should be the same as the model defined in the calibrated cluster model

as the call flow and the path is the same. The only difference with the model is the extra resources such as the number of processors and threads that are available to handle the SIP calls.

For the load balancer, there is double the amount of processors due to the extra machine and four times the load balancer tasks since four load balancer servers are present. For the SIP server, the number of processor has been increased to three as there are three server machines while the number of threads in the server task has increased to sixty reflecting the total number of SIP stack servers. Finally, the number of network processors and tasks has increased to six due to the six different Network Interfaces for the server. This reflects the idea of using different Ethernet interfaces to listen to messages on the same machines to share the load. A full implementation of this model can be found in Appendix B-6. The solved model is shown in Figure 26.

Figure 26: LQN Model of Fully Configured SIP Model

The maximum throughput shown in the model is 1144 calls per second which is close to the measured throughput of 980 calls per second from the actual test runs. From the model, it shows the CPU is fully utilized and identified as the bottleneck of the system. The load balancer's utilization is predicted to be about 20% which is also an acceptable value. With an extra Ethernet interface and server on each machine, it shows that the resources can be fully utilized because the extra Ethernet interface shared the load of the bottleneck.

From the performance modeling and analysis, the bottleneck of the cluster environment was found after the model calibration was done. With the knowledge of the system's behavior, some adjustments were made by adding in another server on the machine by using a different port to avoid the network collision. Using this new setup, the SIP cluster system was able to be fully utilized and was able to handle a larger number of SIP calls.

Chapter 6.0: Conclusions

In this thesis, performance parameters were measured from IBM's SIP application prototype and an LQN performance model was built. This SIP system is a full implementation of the SIP protocol using Java and is deployed on IBM's WebSphere Application Server. The performance tests, where the data was collected, consist of the SIP invite scenario where the user tries to establish a session with another user. Two configurations were studied: one on a standalone environment with a driver and the SIP container, and one on a cluster environment where a load balancer is present in between the two parties from the standalone environment.

In order to understand the behavior of the system, a performance model was constructed using Ethereal captures from different test runs based on the packet flow of the Invite scenario for the SIP Protocol. The model data was collected by examining the packet flow between the parties in the scenario using a common network analysis tool called Ethereal. The service times and throughput rates for the performance model was generated by transforming the packet trace data into an LQN model using the Ethereal parsing algorithm developed here. Both the standalone and cluster configurations were modeled using this method.

The model's predicted throughputs were found to be about double of the measured benchmarks while the CPU utilization of the processors was much lower than expected in the model. It appears that the server is only using one of the four processors found in the system. The model was then calibrated by reducing the number of processors from four to one. Furthermore, the number of threads at the load balancer was

set to one. After all the changes were made, the cluster environment's throughput was still higher than the real system observation. For the cluster environment, it was believed that the network was over saturated as there were too many packets coming through and causing the network not to be able to reply to the system within an acceptable time frame. This caused retransmissions and packet loss due to the failure to reach the time constraint of the protocol implementation. With the use of the recommended output in an open model showing the network being the bottleneck in the system, it seemed to predict the behavior of the system better. After the calibration, the models were able to show that they were able to predict the behavior of the SIP systems.

The performance models show that some of the problems are due to the hardware configuration. One way to fully utilize the machines is to add another SIP server with a different port so the system is able to share the Ethernet load. After adding another server to the SIP container machine, it shows that the SIP servers were able to be fully utilized and able to provide a higher throughput than using only one server.

6.1 Contributions

This thesis makes the following contributions. First, a performance model was built using commonly available network monitoring traces such as Ethereal and a template created from understanding the call flow of the test scenario. This is simpler than the SAME technique and the angio-traces because the structure of the model does not have to be derived. Using Ethereal is a non-intrusive way to measure the system with minimal amount of instrumentation done to the system. The algorithm described in this

this thesis first parsed the packets into a summary file in the form of packet traces. Using these packet traces, they were sorted into different call flow traces containing only traces for a specific call flow. From looking at the interaction patterns, the service times needed are calculated with the timestamps available from these call flow traces. After obtaining data for a few call flows, the average was taken and used as the model parameters.

Another contribution is that the performance model predictions generated by the LQN solver was able to be validated with the real system measurements after some model calibration. This shows that the LQN performance model is suitable to be used to test a real working telecommunication system such as the SIP scenario testing with the use of IBM WAS cluster.

Finally, some research and work was done on the configuration to overcome the bottlenecks that were present in the system. By adding another server to the SIP server machine, more calls were able to be handled and the system was able to be more fully utilized. This is important for capacity planning and performance upgrade as this system can use all the available resources that are present in the setup.

6.2 Future Work

From this thesis, performance modeling on the SIP system was performed from different trace data that were obtained with different tools. The method used was from the extraction of the Ethernet trace capture files from a simple baseline test to calculate the performance parameters. This is one easy way to capture the behavior from the simple call flow since it uses the timestamps from the packets. The thesis shows that this

data is a valid way to create a performance model for the SIP system and can be a reliable way to generate a performance model based on Ethereum. More research should be done to use this kind of data to create a generic performance model for other systems.

Another area to look at for performance analysis in the development cycle is to find an easy way to update the performance model parameters during different points of the development cycle. During development, the behavior of the system changes as more code is added and a new model would be needed to be generated. The system traces should be easily obtained from the module testing of the system, but there needs to be a way to automate the updating process so that models can be easily generated at any point of the development cycle.

Finally, more performance tests should be performed on other functionalities of the system to ensure the performance requirements are satisfied in all areas. Based on the SIPstone document, there are many cases that should be tested for performance other than the one from this thesis. While developing the system, it is important that the performance measures are being taken into consideration.

References

- [Atkins05] S. Atkins, “nmon analyzer – A free tool to produce AIX Performance Report”. DeveloperWorks, IBM, 2005. http://www-128.ibm.com/developerworks/eserver/articles/analyze_aix/
- [Baker04] M. Maker, H. Ong, A. Shafi, “A Study of Java Networking Performance on a Linux Cluster”. Distributed Systems Group, University of Portsmouth, UK, 2004.
- [Bourke01] T. Bourke, “*Server Load Balancing*”. O’Reilly & Associates, Inc., 2001.
- [Carrera03] D. Carrera, J. Guitart, J. Torres, E. Ayguade, J. Labarta, “Complete Instrumentation Requirements for Performance Analysis of Web Based Technologies”. European Center for Parallelism of Barcelona, Computer Architecture Department, Technical University of Catalonia, Spain, 2003.
- [Cisco98] Cisco Systems, Inc, “Cisco 7500 Gigabit Ethernet Interface Processor (GEIP)”. White Paper, 1998.
- [Cisco05] Cisco Systems, Inc, “Ethernet Technologies”. Chapter 7, *Internetworking Technologies Handbook*. 2005. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/index.htm
- [DeMarco05] G. DeMarco, G. Iacovoni, L. Barolli, “A Technique to Analyse Session Initiation Protocol Traffic”. Department of Information and Communication Engineering, Fukuoka Institute of Technology, Japan, 2005.
- [El-Sayed99] H. El-Sayed, “A Framework For Automated Performance Engineering of Distributed Real-Time Systems”, PHD Thesis, Dept. of Systems and computer engineering, Carleton University, 1999.
- [Farrell00] P. Farrell, H. Ong, “Communication Performance over a Gigabit Ethernet Network”. Department of Mathematics and Computer Science, Kent State University, Kent, OH, 2000.
- [Franks05] G. Franks, P. Maly, M. Woodside, D. Petriu, A. Hubbard, “Layered Queuing Network Solver and Simulator User Manual”.

Department of Systems and Computer Engineering, Carleton University, 2005.

- [Gayraud05] R. Gayraud, O. Jacques, "Sipp Reference Documentation v1.0". Technical Paper, 2005.
- [Griffiths05] N. Griffiths, "nmon performance: A free tool to analyze AIX and Linux Performance". DeveloperWorks, IBM, 2005.
- [Gu05] G. Gu, D. Petriu, "From UML to LQN by XML algebra-based model transformations". *Proceedings of 5th ACM Workshop on Software and Performance WOSP'2005*, pp.99-110.
- [Gurbani05] V. Gurbani, L. Jagadeesan, V. Mendiratta, "Characterizing Session Initiation Protocol (SIP) Network Performance and Reliability". Bell Laboratories, Lucent Technologies, Naperville, Illinois, 2005.
- [Hrischuk95] C. Hrishuk, J. Rolia, C.M. Woodside, "Automated generation of software performance model using an object-oriented prototype", *International Workshop on Modeling and Simulation, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '95)*, pp. 399-409, Durham, NC, 1995.
- [Hrishuk95a] Curtis Hrishuk "The automatic generation of software performance models using prototypes". Master's Thesis, Dept. of Systems and Computer Engineering, Carleton University, 1995.
- [IEEE05] IEEE Computer Society, "Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications", IEEE Standard, December 2005.
<http://standards.ieee.org/getieee802/802.3.html>
- [Israr01] T. Israr, "A Lightweight Technique for Extracting Software Architecture and Performance Models from Traces". Master's Thesis, Dept. of Systems and computer engineering, Carleton University, 2001.
- [Israr05] T. Israr, D. Lau, G. Franks, M. Woodside, "Automatic Generation of Layered Queuing Software Performance Models from Commonly Available Tarces". *Proceedings of the First International Workshop on Software and Performance WOSP 05, Palma, Spain*, 2005, pp. 147-158.

- [J2EE] "Understanding Java and the J2EE Platform". Technical Paper, http://media.wiley.com/product_data/excerpt/63/07645396/0764539663.pdf
- [Johnston04] A. Johnston, "*SIP: Understanding the Session Initiation Protocol Second Edition*". Artech House, 2004.
- [Kounev03] S. Kounev, A. Buchmann, "Performance Modeling and Evaluation of Large-Scale J2EE Applications". Department of Computer Science, Darmstadt University of Technology, Germany, 2003.
- [Lazowska84] E. Lazowska, J. Zahorjan, G. Graham, K. Sevcik, "Computer System Analysis Using Queuing Network Models". Prentice-Hall, Inc, Englewood Cliffs, NJ, 1984.
- [Mania02] D. Mania, J. Murphy, J. McManis, "Developing Performance Models from Non-intrusive Monitoring Traces". Performance Engineering Laboratory, Dublin City University, Ireland, 2002.
- [Mata04] K. Mata, J. Pu, J. DeWitt, "Five easy-to-use Performance tools for Linux on PowerPC". DeveloperWorks, IBM, 2004.
- [McCluskey99] G. McCluskey, "Thirty Ways to Improve the Performance of Your Java Programs". Glen McCluskey & Associates LLC. 1999.
- [Menasce94] D. Menasce, V. Almeida, L. Dowdy, "*Capacity Planning and Performance Modeling*". Prentice Hall PRT, Upper Saddle River, NJ, 1994.
- [MGEN] Protean Research Group, "MGEN Tool". Technical Paper, http://cs.itd.nrl.navy.mil/5522/mgen/mgen_index.html
- [Qu00] C. Qu, "J2EE Platform and its Key Technologies". Institut fur Telematik, 2000.
- [Reiss00] S. Reiss, M. Renieris, "Generating Java Trace Data". Department of Computer Science, Brown University, 2000.
- [RFC 1035] RFC 1035, *Domain Names – Implementation and Specification*, Standards, November 1987, <http://www.ietf.org/rfc/rfc1035.txt>
- [RFC 2131] RFC 2131, *Dynamic Host Configuration Protocol*, Standards, March 1997, <http://www.ietf.org/rfc/rfc2131.txt>

- [RFC 2246] RFC 2246, *The TLS Protocol*, Standards, January 1999, <http://www.rfc.net/rfc2246.html>
- [RFC 2327] RFC 2327, *SDP: Session Description Protocol*, Standards, April 1998, <http://www.ietf.org/rfc/rfc2327.txt>
- [RFC 2616] RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1*, Standards, June 1999, <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- [RFC 2821] RFC 2821, *Simple Mail Transfer Protocol*, Standards, April 2001, <http://www.ietf.org/rfc/rfc2821.txt>
- [RFC 2960] RFC 2960, *Stream Control Transmission Protocol*, Standards, October 2000, <http://www.rfc-editor.org/rfc/rfc2960.txt>
- [RFC 3261] RFC 3261, *SIP: Session Initiation Protocol*, Standards, 2002, <http://www.ietf.org/rfc/rfc3261.txt>
- [RFC 3436] RFC 3436, *Transport Layer Security over Stream Control Transmission Protocol*, Standards, December 2002, <http://www.ietf.org/rfc/rfc3436.txt>
- [RFC 3550] RFC 3550, *RTP: A Transport Protocol for Real-Time Applications*, Standards, July 2003. <http://www.ietf.org/rfc/rfc3550.txt>
- [Shaikh01] A. Shaikh, K. Christensen, “Traffic Characteristics of Bulk Data Transfer Using TCP/IP over Gigabit Ethernet”. Department of Computer Science and Engineering, University of South Florida, 2001.
- [SIPP] “SIPP Client”, Technical Paper, <http://sipp.sourceforge.net/>
- [Smith00] C. Smith, L. Williams, “Software Performance AntiPatterns”. *Proceedings 2nd International Workshop on Software and Performance*, 2000.
- [Smith02] C. Smith, L. Williams, “Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software”. *Pearson Education, Inc.*, 2002.
- [Sumimoto98] S. Sumimoto, H. Tezuka, A. Hori, H. Harada, T. Takahashi, Y. Ishikawa, “High Performance Communication using a Gigabit Ethernet”. Tsukuba Research Center, Real World Computing Partnership, Japan, 1998.

- [Suns03] Suns Microsystems. "Java Virtual Machine Profilers Interface (JVMPI)". Technical Paper, 2003.
- [TPROF] "Trace Profiler". Technical Paper, <http://perfinsp.sourceforge.net/>
- [Wang99] J. Wang, S. Keshav. "Efficient and Accurate Ethernet Simulation". Department of Computer Science, Cornell University, 1999.
- [WAS06] "IBM WebSphere Application Server V6.1 delivers flexible, secure infrastructure to provide a reliable foundation for your Service Oriented Architecture". Technical Paper, http://www.ibm.com/common/ssi/rep_ca/6/897/ENUS206-076/ENUS206-076.pdf
- [Woodside05] M. Woodside, G. Franks, "Tutorial Introduction to Layered Modeling of Software Performance". Department of Systems and Computer Engineering, Carleton University, 2005.
- [Wu03] J. Wu, P. Wang, "The Performance Analysis of SIP-T Signaling System in Carrier Class VoIP Network". Graduate Institute of Communication Engineering, NCU, 2003.
- [Zhu03] B. Zhu, "Analysis of SIP in UMTS IP Multimedia Subsystem". Master's Thesis, Graduate Faculty of North Carolina State University, 2003.

APPENDIX A-1: Ethereal Trace Packet with SIP Headers

No.	Time	Source	Destination
25	14.946427	9.42.127.100	9.42.127.128

Protocol Info
Request: INVITE sip:7324200001@perfsipa01 SIP

Frame 25 (378 bytes on wire, 378 bytes captured)
 Ethernet II, Src: Ibm_9c:4a:58 (00:0d:60:9c:4a:58), Dst: Ibm_29:bf:94 (00:11:25:29:bf:94)
 Internet Protocol, Src: 9.42.127.100 (9.42.127.100), Dst: 9.42.127.128 (9.42.127.128)
 User Datagram Protocol, Src Port: 5071 (5071), Dst Port: 5080 (5080)
 Session Initiation Protocol
 Request-Line: INVITE sip:7324200001@perfsipa01 SIP/2.0
 Message Header
 Via: SIP/2.0/UDP
 9.42.127.100:5071;branch=z9hG4bK87asdfsafsf000000001
 To: <sip:7324200001@perfsipa01>
 From: 7323681000<sip:7323681000@9.42.127.100>;tag=12341
 Call-ID: 123412341@9.42.127.100
 CSeq: 1 INVITE
 Contact: <sip:7323681000@9.42.127.100:5071>
 Max-Forwards: 70
 Content-Length: 0

APPENDIX A-2: Ethereal Trace Summary of a Complete Call Flow

```
[Message #] [Call flow ID] [Time Stamp] [Time Stamp of next packet] [Time Stamp Diff] [Source IP] [Dest IP] [SIP Message]
1 < sip:7324200001@perfsipa01> 14.946427 14.947137 0.00071 9.42.127.100 9.42.127.128 SIP Request: INVITE
  sip:7324200001@perfsipa01
2 < sip:7324200001@perfsipa01> 14.947137 14.948150 0.001013 9.42.127.128 9.42.127.100 SIP Status: 100 Trying
3 < sip:7324200001@perfsipa01> 14.948150 14.948263 0.000113 9.42.127.128 9.42.127.100 SIP Request: INVITE
  sip:7324200001@9.42.127.100:5076
4 < sip:7324200001@perfsipa01> 14.948263 14.948690 0.000427 9.42.127.100 9.42.127.128 SIP Status: 300
  Multiple choices
5 < sip:7324200001@perfsipa01> 14.948690 14.948894 0.000204 9.42.127.128 9.42.127.100 SIP Request: ACK
  sip:7324200001@9.42.127.100:5076
6 < sip:7324200001@perfsipa01> 14.948894 14.948986 9.2e-05 9.42.127.128 9.42.127.100 SIP Request: INVITE
  sip:7324200001@9.42.127.100:5074
7 < sip:7324200001@perfsipa01> 14.948986 14.949311 0.000325 9.42.127.100 9.42.127.128 SIP Status: 300
  Multiple choices
8 < sip:7324200001@perfsipa01> 14.949311 14.949410 9.9e-05 9.42.127.128 9.42.127.100 SIP Request: ACK
  sip:7324200001@9.42.127.100:5074
9 < sip:7324200001@perfsipa01> 14.949410 14.949514 0.000104 9.42.127.128 9.42.127.100 SIP Request: INVITE
  sip:7324200001@9.42.127.100:5073
10 < sip:7324200001@perfsipa01> 14.949514 14.950072 0.000558 9.42.127.100 9.42.127.128 SIP Status: 200 OK
11 < sip:7324200001@perfsipa01>;tag=2215435140442218_SIPA.1130104500211.3_1631_1631 14.950072 15.945398
  0.995326 9.42.127.128 9.42.127.100 SIP Status: 200 OK
```

APPENDIX A-3: Ethereal Trace of Retransmission/Failure

```
[Message #] [Call flow ID] [Time Stamp] [Time Stamp of next packet] [Time Stamp Diff] [Source IP] [Dest IP] [SIP Message]
5856 <sip:7324200599@perfsipa01> 0.445123 0.445142 1.9e-05 9.42.127.128 9.42.127.100 SIP Request: INVITE
sip:7324200599@perfsip02:5076
5857 <sip:7324200599@perfsipa01> 0.445142 0.445177 3.5e-05 9.42.127.100 9.42.127.128 SIP Status: 300
Multiple choices
12475 <sip:7324200599@perfsipa01> 1.517995 1.518278 0.000283 9.42.127.128 9.42.127.100 SIP Request: ACK
sip:7324200599@perfsip02:5076
12476 <sip:7324200599@perfsipa01> 1.518278 1.518298 2e-05 9.42.127.128 9.42.127.100 SIP Request: INVITE
sip:7324200599@perfsip02:5074
12477 <sip:7324200599@perfsipa01> 1.518298 1.519092 0.000794 9.42.127.100 9.42.127.128 SIP Status: 300
Multiple choices
21866 <sip:7324200599@perfsipa01> 2.284104 2.284124 2e-05 9.42.127.128 9.42.127.100 SIP Request: INVITE
sip:7324200599@perfsip02:5074
21867 <sip:7324200599@perfsipa01> 2.284124 2.284180 5.6e-05 9.42.127.100 9.42.127.128 SIP Status: 300
Multiple choices
26882 <sip:7324200599@perfsipa01> 3.070826 3.071070 0.000244 9.42.127.128 9.42.127.100 SIP Request: ACK
sip:7324200599@perfsip02:5074
26883 <sip:7324200599@perfsipa01> 3.071070 3.071091 2.1e-05 9.42.127.128 9.42.127.100 SIP Request: INVITE
sip:7324200599@perfsip02:5073
26884 <sip:7324200599@perfsipa01> 3.071091 3.071859 0.000768 9.42.127.100 9.42.127.128 SIP Status: 200 OK
34801 <sip:7324200599@perfsipa01> 3.781068 3.781069 1e-06 9.42.127.128 9.42.127.100 SIP Request: INVITE
sip:7324200599@perfsip02:5073
34819 <sip:7324200599@perfsipa01> 3.781224 3.781225 1e-06 9.42.127.100 9.42.127.128 SIP Status: 200 OK
```

APPENDIX B-1: Original Standalone LQN Model

```
#pragma stop-on-message-loss=false

G "" 1e-006 100 1 0.9 -1

P 3
  p Network f
  p SIPServer f m 4
  p Pdriver f m 4
-1

T 4
  t driver r User -1 Pdriver m 2
  t latency n net -1 Network
  t server n call4 call3 call2 call1 -1 SIPServer m 10
  t driverports n port5074 port5076 port5071 -1 Pdriver
-1

E 9
# ----- driver -----
s User 0.0000001 -1
y User net 11 -1
y User call1 1 -1
y User call2 1 -1
y User call3 1 -1
y User call4 1 -1
#a User 350
# ----- latency -----
s net 0.0000545 -1
# ----- server -----
s call1 0.001345 -1
z call1 port5071 1 -1
s call2 0.000481 -1
z call2 port5076 1 -1
s call3 0.000336 -1
z call3 port5074 1 -1
s call4 0.000433 -1
# ----- driverports -----
s port5071 0.000869 -1
s port5076 0.000178 -1
s port5074 0.000106 -1
-1
```

APPENDIX B-2: Original Cluster LQN Model

```

G "" 1e-006 100 1 0.9 -1

P 4
  p Network f
  p SIPServer f m 4
  p LoadBalancer f m 4
  p Pdriver f m 4
-1

T 7
  t driver r User -1 Pdriver m 20
#  t driver n User -1 Pdriver m 20
  t balancer_down n call4 call3 call2 call1 -1 LoadBalancer i
  t latency n net -1 Network
  t server n server4 server3 server2 server1 -1 SIPServer m 10
  t balancer_up n reply3 reply2 reply1 -1 LoadBalancer i
  t balancer n c4 r3 c3 r2 c2 r1 c1 -1 LoadBalancer m 10
  t driverports n port5074 port5076 port5071 -1 Pdriver
-1

E 23
# ----- driver -----
  s User 1e-007 -1
  y User call1 1 -1
  y User call2 1 -1
  y User call3 1 -1
  y User call4 1 -1
  y User net 22 -1
# a User 290
# ----- balancer_down -----
  s call1 0 -1
  y call1 c1 1 -1
  y call1 server1 1 -1
  s call2 0 -1
  y call2 c2 1 -1
  y call2 server2 1 -1
  s call3 0 -1
  y call3 c3 1 -1
  y call3 server3 1 -1
  s call4 0 -1
  y call4 server4 1 -1
  y call4 c4 1 -1
# ----- latency -----
  s net 0.0000683 -1
# ----- server -----
  s server1 0.00121 -1
  z server1 reply1 1 -1
  s server2 0.000552 -1

```

```

z server2 reply2 1 -1
s server3 0.000428 -1
z server3 reply3 1 -1
s server4 0.000402 -1
# ----- balancer_up -----
s reply1 0 -1
y reply1 r1 1 -1
z reply1 port5071 1 -1
s reply2 0 -1
y reply2 r2 1 -1
z reply2 port5076 1 -1
s reply3 0 -1
y reply3 r3 1 -1
z reply3 port5074 1 -1
# ----- balancer -----
s c1 0.000491 -1
s r1 0.000177 -1
s c2 0.000185 -1
s r2 0.0000927 -1
s c3 0.000176 -1
s r3 0.0000907 -1
s c4 0.000143 -1
# ----- driverports -----
s port5071 0.000514 -1
s port5076 0.000164 -1
s port5074 0.000136 -1
-1

```

APPENDIX B-3: Calibrated Standalone LQN Model

```
#pragma stop-on-message-loss=false

G "" 1e-006 100 1 0.9 -1

P 3
  p Network f
  p SIPServer f
  p Pdriver f m 4
-1

T 4
  t driver r User -1 Pdriver m 10
  #t driver n User -1 Pdriver m 20
  t latency n net -1 Network
  t server n call4 call3 call2 call1 -1 SIPServer m 10
  t driverports n port5074 port5076 port5071 -1 Pdriver
-1

E 9
# ----- driver -----
s User 0.0000001 -1
y User net 11 -1
y User call1 1 -1
y User call2 1 -1
y User call3 1 -1
y User call4 1 -1
#a User 350
# ----- latency -----
s net 0.0000545 -1
# ----- server -----
s call1 0.001345 -1
z call1 port5071 1 -1
s call2 0.000481 -1
z call2 port5076 1 -1
s call3 0.000336 -1
z call3 port5074 1 -1
s call4 0.000433 -1
# ----- driverports -----
s port5071 0.000869 -1
s port5076 0.000178 -1
s port5074 0.000106 -1
-1
```

APPENDIX B-4: Calibrated Cluster LQN Model

```

G "" 1e-006 100 1 0.9 -1

P 4
  p Network f
  p SIPServer f
  p LoadBalancer f m 4
  p Pdriver f m 4
-1

T 7
  t driver r User -1 Pdriver m 20
#  t driver n User -1 Pdriver m 20
  t balancer_down n call4 call3 call2 call1 -1 LoadBalancer i
  t latency n net -1 Network
  t server n server4 server3 server2 server1 -1 SIPServer m 10
  t balancer_up n reply3 reply2 reply1 -1 LoadBalancer i
  t balancer n c4 r3 c3 r2 c2 r1 c1 -1 LoadBalancer
  t driverports n port5074 port5076 port5071 -1 Pdriver
-1

E 23
# ----- driver -----
  s User 1e-007 -1
  y User call1 1 -1
  y User call2 1 -1
  y User call3 1 -1
  y User call4 1 -1
  y User net 22 -1
#  a User 290
# ----- balancer_down -----
  s call1 0 -1
  y call1 c1 1 -1
  y call1 server1 1 -1
  s call2 0 -1
  y call2 c2 1 -1
  y call2 server2 1 -1
  s call3 0 -1
  y call3 c3 1 -1
  y call3 server3 1 -1
  s call4 0 -1
  y call4 server4 1 -1
  y call4 c4 1 -1
# ----- latency -----
  s net 0.0000683 -1
# ----- server -----
  s server1 0.00121 -1
  z server1 reply1 1 -1
  s server2 0.000552 -1

```



```

z server2 reply2 1 -1
s server3 0.000428 -1
z server3 reply3 1 -1
s server4 0.000402 -1
# ----- balancer_up -----
s reply1 0 -1
y reply1 r1 1 -1
z reply1 port5071 1 -1
s reply2 0 -1
y reply2 r2 1 -1
z reply2 port5076 1 -1
s reply3 0 -1
y reply3 r3 1 -1
z reply3 port5074 1 -1
# ----- balancer -----
s c1 0.000491 -1
s r1 0.000177 -1
s c2 0.000185 -1
s r2 0.0000927 -1
s c3 0.000176 -1
s r3 0.0000907 -1
s c4 0.000143 -1
# ----- driverports -----
s port5071 0.000514 -1
s port5076 0.000164 -1
s port5074 0.000136 -1
-1

```

APPENDIX B-5: Open Calibrated Cluster LQN Model

```

G "" 1e-006 100 1 0.9 -1

P 4
  p Network f
  p SIPServer f
  p LoadBalancer f m 4
  p Pdriver f m 4
-1

T 7
  #t driver r User -1 Pdriver m 2
  t driver n User -1 Pdriver m 20
  t balancer_down n call4 call3 call2 call1 -1 LoadBalancer i
  t latency n net -1 Network
  t server n server4 server3 server2 server1 -1 SIPServer m 10
  t balancer_up n reply3 reply2 reply1 -1 LoadBalancer i
  t balancer n c4 r3 c3 r2 c2 r1 c1 -1 LoadBalancer
  t driverports n port5074 port5076 port5071 -1 Pdriver
-1

E 23
# ----- driver -----
s User 1e-007 -1
y User call1 1 -1
y User call2 1 -1
y User call3 1 -1
y User call4 1 -1
y User net 22 -1
a User 290
# ----- balancer_down -----
s call1 0 -1
y call1 c1 1 -1
y call1 server1 1 -1
s call2 0 -1
y call2 c2 1 -1
y call2 server2 1 -1
s call3 0 -1
y call3 c3 1 -1
y call3 server3 1 -1
s call4 0 -1
y call4 server4 1 -1
y call4 c4 1 -1
# ----- latency -----
s net 0.0000683 -1
# ----- server -----
s server1 0.00121 -1
z server1 reply1 1 -1

```

```

s server2 0.000552 -1
z server2 reply2 1 -1
s server3 0.000428 -1
z server3 reply3 1 -1
s server4 0.000402 -1
# ----- balancer_up -----
s reply1 0 -1
y reply1 r1 1 -1
z reply1 port5071 1 -1
s reply2 0 -1
y reply2 r2 1 -1
z reply2 port5076 1 -1
s reply3 0 -1
y reply3 r3 1 -1
z reply3 port5074 1 -1
# ----- balancer -----
s c1 0.000491 -1
s r1 0.000177 -1
s c2 0.000185 -1
s r2 0.0000927 -1
s c3 0.000176 -1
s r3 0.0000907 -1
s c4 0.000143 -1
# ----- driverports -----
s port5071 0.000514 -1
s port5076 0.000164 -1
s port5074 0.000136 -1
-1

```

APPENDIX B-6: LQN model for Full Configuration

```

G "" 1e-006 100 1 0.9 -1

P 4
  p Network f m 6
  p SIPServer f m 3
  p LoadBalancer f m 8
  p Pdriver f m 8
-1

T 7
  t driver r User -1 Pdriver m 20
#  t driver n User -1 Pdriver m 20
  t balancer_down n call4 call3 call2 call1 -1 LoadBalancer i
  t latency n net -1 Network m 6
  t server n server4 server3 server2 server1 -1 SIPServer m 60
  t balancer_up n reply3 reply2 reply1 -1 LoadBalancer i
  t balancer n c4 r3 c3 r2 c2 r1 c1 -1 LoadBalancer m 4
  t driverports n port5074 port5076 port5071 -1 Pdriver m 4
-1

E 23
# ----- driver -----
s User 1e-007 -1
y User call1 1 -1
y User call2 1 -1
y User call3 1 -1
y User call4 1 -1
y User net 22 -1
# a User 290
# ----- balancer_down -----
s call1 0 -1
y call1 c1 1 -1
y call1 server1 1 -1
s call2 0 -1
y call2 c2 1 -1
y call2 server2 1 -1
s call3 0 -1
y call3 c3 1 -1
y call3 server3 1 -1
s call4 0 -1
y call4 server4 1 -1
y call4 c4 1 -1
# ----- latency -----
s net 0.0000683 -1
# ----- server -----
s server1 0.00121 -1
z server1 reply1 1 -1

```

```

s server2 0.000552 -1
z server2 reply2 1 -1
s server3 0.000428 -1
z server3 reply3 1 -1
s server4 0.000402 -1
# ----- balancer_up -----
s reply1 0 -1
y reply1 r1 1 -1
z reply1 port5071 1 -1
s reply2 0 -1
y reply2 r2 1 -1
z reply2 port5076 1 -1
s reply3 0 -1
y reply3 r3 1 -1
z reply3 port5074 1 -1
# ----- balancer -----
s c1 0.000491 -1
s r1 0.000177 -1
s c2 0.000185 -1
s r2 0.0000927 -1
s c3 0.000176 -1
s r3 0.0000907 -1
s c4 0.000143 -1
# ----- driverports -----
s port5071 0.000514 -1
s port5076 0.000164 -1
s port5074 0.000136 -1
-1

```

APPENDIX C-1: Ethereal Trace Parsing Script – analyze.sh

```
#!/bin/bash

if [ -z $1 ]
then
    echo [!] ERROR: Input filename required!
    exit
fi

if [ ! -e $1 ]
then
    echo [!] ERROR: File $1 Not Found!
    exit
fi

# Create suitable file names

echo $1 > name.tmp
sed -e 's/.out//g' name.tmp > file.tmp
filenameAll=`cat file.tmp`All.out

# Generating trace file
echo [*] Generating traces for $1
awk '
BEGIN {packet=0;
count=1;
#print "Packet ID Time Source Dest Protocol Info"
}
{ if ($1 == "No.") {packet++; count=0};
if (count == 1) tempstring = (" " $2 " " $3 " " $4 " " $5 " " $6 " " $7
" " $8 " " $9);
if ($1 == "To:") {printf packet " " $2; print tempstring};
count++;}' $1 >> $filenameAll

# Create trace files
echo [*] Sorting traces with timestamp calculations

# Getting files for all packets with timestamps
filename=`cat file.tmp`time.out
awk '
BEGIN {first=1;
prevtime=0;
#print "Packet ID Time Source Dest Protocol Info"
}
{
packet++;
if (first == 0) {
diff=$3-prevtime;
```

```
    print tempstring $3 " " diff tempstring2};
tempstring=($1 " " $2 " " $3 " ");
tempstring2=(" " $4 " " $5 " " $6 " " $7 " " $8 " " $9 " " $10 " " $11
" " $12);
prevtime=$3;
first=0;
}
END { print tempstring " N/A " tempstring2}' $filenameAll >> $filename
```

```
# Cleanup temporary files
```

```
rm name.tmp
rm file.tmp
```

APPENDIX C-2: Ethereal Trace Parsing Script – packet.sh

```

#!/bin/bash

if [ -z $1 ]
then
    echo [!] ERROR: Input filename required!
    exit
fi

if [ ! -e $1 ]
then
    echo [!] ERROR: File $1 Not Found!
    exit
fi

if [ -z $2 ]
then
    echo [!] ERROR: Packet ID required!
    exit
fi

# Create suitable file names

echo $1 > name.tmp
filetest=`grep -c time.out name.tmp`
if [ $filetest -eq 0 ]
then
    echo [!] ERROR: Need the parsed result file as input! ie.
    XXXXXtime.out
    exit
fi
sed -e 's/time.out//g' name.tmp > file.tmp
filename=`cat file.tmp`$2.out
results=`cat file.tmp`Perf.out

# Extract the packet from the trace file
echo Extracting packet and performing analysis

grep $2 $1 > $filename

# Extract the performance number for the packet and put it in the
results file

awk '
BEGIN {
count=1;
error=0;
}

```



```

{
if (count == 1 && $10 == "INVITE") {
    p1 = $3;
    count++;
}
else if (count == 2 && $10 == "100") {
    p2 = $3;
    count++;
}
else if (count == 3 && $10 == "INVITE") {
    p3 = $3;
    count++;
}
else if (count == 4 && $10 == "300") {
    p4 = $3;
    count++;
}
else if (count == 5 && $10 == "ACK") {
    p5 = $3;
    count++;
}
else if (count == 6 && $10 == "INVITE") {
    p6 = $3;
    count++;
}
else if (count == 7 && $10 == "300") {
    p7 = $3;
    count++;
}
else if (count == 8 && $10 == "ACK") {
    p8 = $3;
    count++;
}
else if (count == 9 && $10 == "INVITE") {
    p9 = $3;
    count++;
}
else if (count == 10 && $10 == "200") {
    p10 = $3;
    count++;
}
else if (count == 11 && $10 == "200") {
    p11 = $3;
    count++;
}
else {
    error=1;
}
}
END {
if (count == 12) {
    #print p1 " " p2 " " p3 " " p4 " " p5 " " p6 " " p7 " " p8 " " p9 " "
    p10 " " p11;
    clto2 = (p2 - p1)*1000;
}
}

```

```

c1to3 = (p3 - p1)*1000;
c2to3 = (p3 - p2)*1000;
c4to5 = (p5 - p4)*1000;
c4to6 = (p6 - p4)*1000;
c5to6 = (p6 - p5)*1000;
c7to8 = (p8 - p7)*1000;
c7to9 = (p9 - p7)*1000;
c8to9 = (p9 - p8)*1000;
c10to11 = (p11 - p10)*1000;
output = c1to2 " " c1to3 " " c2to3 " " c4to5 " " c4to6 " " c5to6 " "
c7to8 " " c7to9 " " c8to9 " " c10to11;
if (error == 1) {
    output = output " RETRANSMISSION";
}
print output;
}
else {
    print "ERROR1";
}
}' $filename >> $results

# Checking for error in the output
warning=`grep -c RETRANSMISSION $results`
if [ $warning -ge 1 ]
then
    echo [!] Warning: Retransmission detected!
fi

error1=`grep -c ERROR1 $results`
if [ $error1 -ge 1 ]
then
    echo [*] ERROR: Call did not complete!!!!
fi

# Move packet capture file to packet directory
mv $filename packet

# Cleanup temporary files

rm name.tmp
rm file.tmp

```

APPENDIX C-3: Ethereal Trace Parsing Script – packetSLSP.sh

```

#!/bin/bash

if [ -z $1 ]
then
    echo [!] ERROR: Input filename required!
    exit
fi

if [ ! -e $1 ]
then
    echo [!] ERROR: File $1 Not Found!
    exit
fi

if [ -z $2 ]
then
    echo [!] ERROR: Packet ID required!
    exit
fi

# Create suitable file names

echo $1 > name.tmp
filetest=`grep -c time.out name.tmp`
if [ $filetest -eq 0 ]
then
    echo [!] ERROR: Need the parsed result file as input! ie.
    XXXXXtime.out
    exit
fi
sed -e 's/time.out//g' name.tmp > file.tmp
filename=`cat file.tmp`$2.out
results=`cat file.tmp`Perf.out

# Extract the packet from the trace file
echo Extracting packet and performing analysis

grep $2 $1 > $filename

# Extract the performance number for the packet and put it in the
results file

awk '
BEGIN {
count=1;
error=0;
}

```

```

{
if (count == 1 && $10 == "INVITE") {
    p1 = $3;
    count++;
}
else if (count == 2 && $10 == "INVITE") {
    r1 = $3;
    count++;
}
else if (count == 3 && $10 == "100") {
    p2 = $3;
    count++;
}
else if (count == 4 && $10 == "100") {
    r2 = $3;
    count++;
}
else if (count == 5 && $10 == "INVITE") {
    p3 = $3;
    count++;
}
else if (count == 6 && $10 == "INVITE") {
    r3 = $3;
    count++;
}
else if (count == 7 && $10 == "300") {
    p4 = $3;
    count++;
}
else if (count == 8 && $10 == "300") {
    r4 = $3;
    count++;
}
else if (count == 9 && $10 == "ACK") {
    p5 = $3;
    count++;
}
else if (count == 10 && $10 == "ACK") {
    r5 = $3;
    count++;
}
else if (count == 11 && $10 == "INVITE") {
    p6 = $3;
    count++;
}
else if (count == 12 && $10 == "INVITE") {
    r6 = $3;
    count++;
}
else if (count == 13 && $10 == "300") {
    p7 = $3;
    count++;
}
else if (count == 14 && $10 == "300") {

```

```

    r7 = $3;
    count++;
}
else if (count == 15 && $10 == "ACK") {
    p8 = $3;
    count++;
}
else if (count == 16 && $10 == "ACK") {
    r8 = $3;
    count++;
}
else if (count == 17 && $10 == "INVITE") {
    p9 = $3;
    count++;
}
else if (count == 18 && $10 == "INVITE") {
    r9 = $3;
    count++;
}
else if (count == 19 && $10 == "200") {
    p10 = $3;
    count++;
}
else if (count == 20 && $10 == "200") {
    r10 = $3;
    count++;
}
else if (count == 21 && $10 == "200") {
    p11 = $3;
    count++;
}
else if (count == 22 && $10 == "200") {
    r11 = $3;
    count++;
}
else {
    print count;
    error=1;
}
}
END {
if (count == 23) {
    #print p1 " " p2 " " p3 " " p4 " " p5 " " p6 " " p7 " " p8 " " p9 " "
    p10 " " p11;
    c1to3 = (r1 - p1)*1000 + (r3 - p3)*1000;
    cp2tor2 = (r2 - p2)*1000;

    c4to6 = (r4 - p4)*1000 + (r6 - p6)*1000;
    cp5tor5 = (r5 - p5)*1000;

    c7to9 = (r7 - p7)*1000 + (r9 - p9)*1000;
    cp8tor8 = (r8 - p8)*1000;

    cp10tor11 = (r10 - p10)*1000 + (r11 - p11)*1000;

```

```

    output = clto3 " " cp2tor2 " " c4to6 " " cp5tor5 " " c7to9 " "
cp8tor8 " " cp10tor11;
    if (error == 1) {
        output = output " RETRANSMISSION";
    }
    print output;
}
else {
    print "ERROR1";
}
}' $filename >> $results

# Checking for error in the output
warning=`grep -c RETRANSMISSION $results`
if [ $warning -ge 1 ]
then
    echo [!] Warning: Retransmission detected!
fi

error1=`grep -c ERROR1 $results`
if [ $error1 -ge 1 ]
then
    echo [*] ERROR: Call did not complete!!!!
fi

# Move packet capture file to packet directory
mv $filename packet

# Cleanup temporary files

rm name.tmp
rm file.tmp

```

Appendix D-1: Original Standalone LQN Model Output

Generated by lqns, version 3.10

Copyright the Real-Time and Distributed Systems Group,
Department of Systems and Computer Engineering
Carleton University, Ottawa, Ontario, Canada. K1S 5B6

Input: sip_closed_ORG.lqn
Output: sip_closed_ORG.out
Command line: lqns -p
Mon May 22 12:06:16 2006

#pragma stop-on-message-loss = false

Convergence test value: 7.34147e-007
Number of iterations: 4

MVA solver information:

Submdl	n	k	srv	step()	mean	stddev	wait()
mean				Elapsed			
1	4	1	2	29	7.25	3.3448	765
191.25				0:00:00.00			
2	7	2	3	52	7.4286	1.0498	3152
450.29				0:00:00.00			
3	4	2	1	31	7.75	1.299	1482
370.5				0:00:00.00			
Total	15	0	0	112	7.4667	1.9956	5399
359.93				0:00:00.00			

Elapsed: 0:00:00.00

Processor identifiers and scheduling algorithms:

Processor Name	Type	Copies	Scheduling	
Network	Uni	1	FCFS	(HVFCFS_Server)
SIPServer	Mult(4)	1	FCFS	(Conway_Multi_Server)
Pdriver	Mult(4)	1	FCFS	(Conway_Multi_Server)

Task information:

Task Name	Type	Copies	Processor Name	Pri	Entry List
driver	ref(2)	1	Pdriver	0	User
latency	serv	1	Network	0	net
(FCFS_Server)					
server	mult(10)	1	SIPServer	0	call4, call3,
call2, call1					(Conway_Multi_Server)

driverports serv 1 Pdriver 0 port5074,
port5076, port5071(HVFCFS_Server)

Entry execution demands:

Task Name	Entry Name	Phase 1
driver	User	1e-007
latency	net	5.45e-005
server	call4	0.000433
	call3	0.000336
	call2	0.000481
	call1	0.001345
driverports	port5074	0.000106
	port5076	0.000178
	port5071	0.000869

Mean number of rendezvous from entry to entry:

Task Name	Source Entry	Target Entry	Phase 1
driver	User	net	11
	User	call1	1
	User	call2	1
	User	call3	1
	User	call4	1

Mean number of non-blocking sends from entry to entry:

Task Name	Source Entry	Target Entry	Phase 1
server	call3	port5074	1
	call2	port5076	1
	call1	port5071	1

Phase type flags:

All phases are stochastic.

Squared coefficient of variation of execution segments:

All executable segments are exponential.

Open arrival rates per entry:

All open arrival rates are 0.

Type 1 throughput bounds:

Task Name	Entry Name	Throughput
driver	User	626.056
latency	net	18348.6

server	call4	23094.7
	call3	29761.9
	call2	20790
	call1	7434.94
driverports	port5074	9433.96
	port5076	5617.98
	port5071	1150.75

Mean delay for a rendezvous:

Task Name	Source Entry	Target Entry	Phase 1
driver	User	net	1.02275e-005
	User	call1	0
	User	call2	0
	User	call3	0
	User	call4	0

Mean delay for a send-no-reply request:

Task Name	Source Entry	Target Entry	Phase 1
server	call3	port5074	0.000582319 (Async
Call)			
	call2	port5076	0.000715596 (Async
Call)			
	call1	port5071	0.00357464 (Async
Call)			

Service times:

Task Name	Entry Name	Phase 1
driver	User	0.0033071
latency	net	5.45e-005
server	call4	0.000433
	call3	0.000336
	call2	0.000481
	call1	0.001345
driverports	port5074	0.000106
	port5076	0.000178
	port5071	0.000869

Service time variance (per phase)
and squared coefficient of variation (over all phases):

Task Name	Entry Name	Phase 1	coeff of var **2
driver	User	7.60934e-006	0.695748
latency	net	2.97025e-009	1
server	call4	1.87489e-007	1
	call3	1.12896e-007	1
	call2	2.31361e-007	1
	call1	1.80903e-006	1

driverports	port5074	1.1236e-008	1
	port5076	3.1684e-008	1
	port5071	7.55161e-007	1

Throughputs and utilizations per phase:

Task Name	Entry Name	Throughput	Phase 1	Total
driver	User	604.759	2	2
latency	net	6652.35	0.362553	0.362553
server	call4	604.759	0.261861	0.261861
	call3	604.759	0.203199	0.203199
	call2	604.759	0.290889	0.290889
	call1	604.759	0.813401	0.813401
	Total	2419.04	1.56935	1.56935
driverports	port5074	604.759	0.0641045	0.0641045
	port5076	604.759	0.107647	0.107647
	port5071	604.759	0.525536	0.525536
	Total	1814.28	0.697287	0.697287

Utilization and waiting per phase for processor: Network

Task Name	Pri	n	Entry Name	Utilization	Ph1	wait
latency	0	1	net	0.362553	0	

Utilization and waiting per phase for processor: SIPServer

Task Name	Pri	n	Entry Name	Utilization	Ph1	wait
server	0	10	call4	0.261861	0	
			call3	0.203199	0	
			call2	0.290889	0	
			call1	0.813401	0	
server			Total	1.56935		

Utilization and waiting per phase for processor: Pdriver

Task Name	Pri	n	Entry Name	Utilization	Ph1	wait
driver	0	2	User	6.04759e-005	0	
driverports	0	1	port5074	0.0641045	0	
			port5076	0.107647	0	
			port5071	0.525536	0	
driverports			Total	0.697287		
Total processor utilization:				0.697348		

Appendix D-2: Original Cluster LQN Model Output

Generated by lqns, version 3.9

Copyright the Real-Time and Distributed Systems Group,
Department of Systems and Computer Engineering
Carleton University, Ottawa, Ontario, Canada. K1S 5B6

Input: SIPcluster_closed_ORG.lqn
Output: SIPcluster_closed_ORG.out
Command line: lqns -p
Wed May 17 19:47:49 2006

Convergence test value: 6.5225e-007
Number of iterations: 6

MVA solver information:

Submdel	n	k	srv	step()	mean	stddev	wait()
mean		stddev		Elapsed			
1	6	1	2	43	7.1667	4.0173	1215
202.5		254.26		0:00:00.00			
2	11	2	2	89	8.0909	2.6442	6376
579.64		478.78		0:00:00.00			
3	11	1	2	77	7	2.8604	1258
114.36		125.96		0:00:00.00			
4	11	2	2	84	7.6364	1.4938	3996
363.27		167.6		0:00:00.00			
5	6	3	2	62	10.333	2.9814	10410
1735		1162.8		0:00:00.00			
Total	45	0	0	355	7.8889	2.9456	23255
516.78		716.25		0:00:00.00			

Elapsed: 0:00:00.00

Processor identifiers and scheduling algorithms:

Processor Name	Type	Copies	Scheduling
Network	Uni	1	FCFS (HVFCFS_Server)
SIPServer	Mult(4)	1	FCFS (Conway_Multi_Server)
LoadBalancer	Mult(4)	1	FCFS (Conway_Multi_Server)
Pdriver	Mult(4)	1	FCFS (Conway_Multi_Server)

Task information:

Task Name	Type	Copies	Processor Name	Pri	Entry List
driver	ref(20)	1	Pdriver	0	User

balancer_down	inf	1	LoadBalancer	0	call4, call3,
call2, call1	(Delay_Server)				
latency	serv	1	Network	0	net
(FCFS_Server)					
server	mult(10)	1	SIPServer	0	server4, server3,
server2, server1	(Conway_Multi_Server)				
balancer_up	inf	1	LoadBalancer	0	reply3, reply2,
reply1	(Delay_Server)				
balancer	mult(10)	1	LoadBalancer	0	c4, r3, c3, r2, c2,
r1, c1	(Conway_Multi_Server)				
driverports	serv	1	Pdriver	0	port5074, port5076,
port5071	(HVFCFS_Server)				

Entry execution demands:

Task Name	Entry Name	Phase 1
driver	User	1e-007
balancer_down	call4	0
	call3	0
	call2	0
	call1	0
latency	net	6.83e-005
server	server4	0.000402
	server3	0.000428
	server2	0.000552
	server1	0.00121
balancer_up	reply3	0
	reply2	0
	reply1	0
balancer	c4	0.000143
	r3	9.07e-005
	c3	0.000176
	r2	9.27e-005
	c2	0.000185
	r1	0.000177
	c1	0.000491
driverports	port5074	0.000136
	port5076	0.000164
	port5071	0.000514

Mean number of rendezvous from entry to entry:

Task Name	Source Entry	Target Entry	Phase 1
driver	User	call1	1
	User	call2	1
	User	call3	1
	User	call4	1
	User	net	22
balancer_down	call4	server4	1
	call4	c4	1
	call3	c3	1

	call3	server3	1
	call2	c2	1
	call2	server2	1
	call1	c1	1
	call1	server1	1
balancer_up	reply3	r3	1
	reply2	r2	1
	reply1	r1	1

Mean number of non-blocking sends from entry to entry:

Task Name	Source Entry	Target Entry	Phase 1
server	server3	reply3	1
	server2	reply2	1
	server1	reply1	1
balancer_up	reply3	port5074	1
	reply2	port5076	1
	reply1	port5071	1

Phase type flags:

All phases are stochastic.

Squared coefficient of variation of execution segments:

All executable segments are exponential.

Open arrival rates per entry:

All open arrival rates are 0.

Type 1 throughput bounds:

Task Name	Entry Name	Throughput
driver	User	3929.5
balancer_down	call4	1834.86
	call3	1655.63
	call2	1356.85
	call1	587.889
latency	net	14641.3
server	server4	24875.6
	server3	23364.5
	server2	18115.9
	server1	8264.46
balancer_up	reply3	11025.4
	reply2	10787.5
	reply1	5649.72
balancer	c4	69930.1
	r3	110254
	c3	56818.2
	r2	107875
	c2	54054.1

	r1	56497.2
	c1	20366.6
driverports	port5074	7352.94
	port5076	6097.56
	port5071	1945.53

Mean delay for a rendezvous:

Task Name	Source Entry	Target Entry	Phase 1
driver	User	call1	0
	User	call2	0
	User	call3	0
	User	call4	0
	User	net	0.00113228
balancer_down	call4	server4	1.82516e-010
	call4	c4	8.03209e-013
	call3	c3	8.03209e-013
	call3	server3	1.82516e-010
	call2	c2	8.03209e-013
	call2	server2	1.82516e-010
	call1	c1	8.03209e-013
	call1	server1	1.82516e-010
balancer_up	reply3	r3	0
	reply2	r2	0
	reply1	r1	0

Mean delay for a send-no-reply request:

Task Name	Source Entry	Target Entry	Phase 1
server Call)	server3	reply3	9.07e-005 (Async
Call)	server2	reply2	9.27e-005 (Async
Call)	server1	reply1	0.000177 (Async
balancer_up Call)	reply3	port5074	0.000336617 (Async
Call)	reply2	port5076	0.000382911 (Async
Call)	reply1	port5071	0.00124971 (Async

Service times:

Task Name	Entry Name	Phase 1
driver	User	0.0300552
balancer_down	call4	0.00055884
	call3	0.00061784
	call2	0.00075084
	call1	0.00171484
latency	net	6.83e-005

server	server4	0.00041578
	server3	0.00044178
	server2	0.00056578
	server1	0.00122378
balancer_up	reply3	9.07e-005
	reply2	9.27e-005
	reply1	0.000177
balancer	c4	0.00014306
	r3	9.07e-005
	c3	0.00017606
	r2	9.27e-005
	c2	0.00018506
	r1	0.000177
	c1	0.00049106
driverports	port5074	0.000136
	port5076	0.000164
	port5071	0.000514

Service time variance (per phase)
and squared coefficient of variation (over all phases):

Task Name	Entry Name	Phase 1	coeff of var **2
driver	User	0.000773517	0.856311
balancer_down	call4	5.68922e-007	1.8217
	call3	6.66684e-007	1.7465
	call2	1.04783e-006	1.85864
	call1	5.18293e-006	1.76249
latency	net	4.66489e-009	1
server	server4	1.61794e-007	0.935911
	server3	1.83374e-007	0.939562
	server2	3.04894e-007	0.952475
	server1	1.46429e-006	0.977733
balancer_up	reply3	2.46795e-008	3
	reply2	2.57799e-008	3
	reply1	9.3987e-008	3
balancer	c4	2.0449e-008	0.99916
	r3	8.22649e-009	1
	c3	3.0976e-008	0.999317
	r2	8.59329e-009	1
	c2	3.4225e-008	0.999351
	r1	3.1329e-008	1
	c1	2.41081e-007	0.999755
driverports	port5074	1.8496e-008	1
	port5076	2.6896e-008	1
	port5071	2.64196e-007	1

Throughputs and utilizations per phase:

Task Name	Entry Name	Throughput	Phase 1	Total
driver	User	665.443	20	20
balancer_down	call4	665.443	0.371876	0.371876
	call3	665.443	0.411138	0.411138

	call2	665.443	0.499642	0.499642
	call1	665.443	1.14113	1.14113
	Total	2661.77	2.42378	2.42378
latency	net	14639.7	0.999895	0.999895
server	server4	665.443	0.276678	0.276678
	server3	665.443	0.293979	0.293979
	server2	665.443	0.376494	0.376494
	server1	665.443	0.814356	0.814356
	Total	2661.77	1.76151	1.76151
balancer_up	reply3	665.443	0.0603557	0.0603557
	reply2	665.443	0.0616866	0.0616866
	reply1	665.443	0.117783	0.117783
	Total	1996.33	0.239826	0.239826
balancer	c4	665.443	0.0951983	0.0951983
	r3	0	0	0
	c3	665.443	0.117158	0.117158
	r2	0	0	0
	c2	665.443	0.123147	0.123147
	r1	0	0	0
	c1	665.443	0.326773	0.326773
	Total	2661.77	0.662276	0.662276
driverports	port5074	665.443	0.0905003	0.0905003
	port5076	665.443	0.109133	0.109133
	port5071	665.443	0.342038	0.342038
	Total	1996.33	0.541671	0.541671

Utilization and waiting per phase for processor: Network

Task Name	Pri	n	Entry Name	Utilization	Ph1	wait
latency	0	1	net	0.999895	0	

Utilization and waiting per phase for processor: SIPServer

Task Name	Pri	n	Entry Name	Utilization	Ph1	wait
server	0	10	server4	0.267508	1.37801e-005	
			server3	0.28481	1.37801e-005	
			server2	0.367325	1.37801e-005	
			server1	0.805186	1.37801e-005	
server		Total		1.72483		

Utilization and waiting per phase for processor: LoadBalancer

Task Name	Pri	n	Entry Name	Utilization	Ph1	wait
balancer_down	0	1	call4	0	0	
			call3	0	0	
			call2	0	0	
			call1	0	0	
balancer_down		Total		0		
balancer_up	0	1	reply3	0	0	
			reply2	0	0	
			reply1	0	0	

balancer_up	Total		0	
balancer	0 10	c4	0.0951584	6.0103e-008
		r3	0	0
		c3	0.117118	6.0103e-008
		r2	0	0
		c2	0.123107	6.0103e-008
		r1	0	0
		c1	0.326733	6.0103e-008
balancer	Total		0.662116	
Total processor utilization:			0.662116	

Utilization and waiting per phase for processor: Pdriver

Task Name	Pri n	Entry Name	Utilization	Ph1 wait
driver	0 20	User	6.65443e-005	1.09234e-011
driverports	0 1	port5074	0.0905002	3.03431e-013
		port5076	0.109133	3.03431e-013
		port5071	0.342038	3.03431e-013
driverports	Total		0.541671	
Total processor utilization:			0.541737	