EVALUATING IP MULTIMEDIA SUBSYSTEM PERFORMANCE

Minh Nhat Bui

A thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science
Concordia University
Montréal, Québec, Canada

September 2014

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Minh Nhat Bui**

Entitled: **Evaluating IP Multimedia Subsystem Performance**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

______________________________ Chair
Dr. M. Zahangir Kabir

______________________________ Examiner
Dr. Abdelwahab Hamou-Lhadj

______________________________ Examiner
Dr. Roch Glitho

______________________________ Co-supervisor
Dr. Anjali Agarwal

______________________________ Co-supervisor
Dr. J. William Atwood

Approved ______________________________
Chair of Department or Graduate Program Director

__________ 20 ______ ______________________________

Dr. Amir Asif, Dean

Faculty of Engineering and Computer Science

Abstract

Evaluating IP Multimedia Subsystem Performance

Minh Nhat Bui

The IP Multimedia Subsystem has seen increasing deployment over the past few years. This also means that the number of subscribers has significantly increased. Thus IMS performance analysis becomes a critical area to be researched. There are various methods to conduct the system evaluation. Most of the previous work has concentrated solely on methodology models or physical system measurements. This thesis proposes a new model combining a queueing network model and physical system measurements to achieve a precise and realistic evaluation result.

The proposed model uses the concept of an open multi-class queueing network with heterogeneous requests. The requests enter IMS system at a Poisson distribution rate and are grouped into different classes. They travel within the queueing network with or without changing classes depending on which nodes are being processed. After passing through the required nodes, the requests exit the network. The serving time at each node is measured and the final system response time can then be derived using our formula. Our model can also predict the IMS saturation point over which the system becomes unstable. In addition, the CPU utilizations for each IMS component are derived. Based on these values, multiple IMS components can be efficiently grouped onto one machine to save system resource.

The proposed model is verified for the IMS registration, call setup and termination procedures in an IMS test-bed system. The measured and calculated performance results match precisely. In addition, the model can group multiple IMS components or can separate one IMS component into multiple logical components. This scalability is crucial in production systems where the number of components grows continuously. Service providers can use this model to study the message flows in their IMS network and see how the system responds when the traffic load changes.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisors Professor J. William Atwood and Professor Anjali Agarwal for their continuous support throughout the process. Without their assistance and dedicated involvement, this thesis would have never been accomplished.

Besides, I would like to thank Dr. Mama Nsangou Mouchili for his valuable suggestions. His extensive academic and practical knowledge helped me overcome many obstacles I encountered throughout the process.

Achieving up to this point requires more than academic support; I would like to thank my parents who supported and listened to me during the past two years. I also would like to thank my grandmother, my uncle Bon Tran and my aunt Anh Huynh who opened both their home and heart to me when I first arrived in the city. For many memorable evenings out and in, I must thank my cousins Susan and Viviane.

Last but not least, I would like to thank my friend Chloe Luc who is always there cheering me up and standing by me through good and bad. She has been kind and supportive to me over the time. Every time I was ready to quit, she did not let me and I am forever grateful.

Once again, thank you everyone. This thesis stands as a statement to your love and encouragement.

Contents

List of Figures **viii**

List of Tables **x**

List of Acronyms **xi**

1 Introduction **1**

2 Background **4**

2.1 IP Multimedia Subsystem (IMS) Architecture 4

2.1.1 Call Session Control Function (CSCF) 7

2.1.2 Home Subscriber Server (HSS) 10

2.1.3 User Identity 10

2.2 Protocols in IMS 12

2.2.1 Session Initiation Protocol (SIP) 12

2.2.2 Diameter Protocol 15

2.3 Registration Procedure in IMS 15

2.4 Queueing Network 20

2.4.1 Queue Definition 20

2.4.2 Queueing Network 23

3 Analysis **26**

3.1 Related Work 26

3.2 Proposed Analysis 31

3.2.1 Registration Procedure Analysis Model 31

3.2.2 Mathematical Analysis 35

3.2.3 Call Setup and Termination Analysis 38

4 Testing and Evaluating **40**
4.1 Introduction to Test-bed Environment 40
4.2 OpenIMS Implementation 42
4.3 IMS Functional Test 42
4.4 IMS Bench SIPp Implementation 43
4.4.1 Component Structure 43
4.4.2 Installation 45
4.4.3 Test Cases 46
4.5 Measured Performance Results 50
4.6 Response Time Evaluation 51
4.6.1 Test Case 1: Registration Procedure Only 51
4.6.2 Test Case 2: Registration, Call Setup and Termination Procedure . 60
4.7 System Performance Evaluation 66
4.7.1 System Saturation Point 66
4.7.2 System CPU Utilization 73

5 Conclusion and Future Work **76**

Appendix **78**
A Scripts to Start IMS Components 78
B Call Setup and Termination Procedure 81
C Open Source IMS Implementation Steps 83
C.1 Prerequisite Packages 83
C.2 OpenIMS Core Source Code 84
C.3 Configure DNS Server 85
C.4 Configure DNS Core Components 90
C.5 Start IMS Components 95
D IMS Bench SIPp Implementation Steps 98
E Database Structure 101
F myMONSTER Installation 103
G Adjust Ubuntu Kernel Timer Frequency 104
H Script to Generate User List 106
I XML Configuration For SIPp Manager 108
J Scripts to Add Users to FHoSS 112
K Test-bed Specific Configurations and Scenarios 117
L How to Collect IMS Bench SIPp Results 121

M Matlab Code to Calculate Arrival Rate 122
N Matlab Code to Calculate Saturation Points 123
O Matlab Code to Calculate CPU Utilization 126
P Matlab Code to Calculate Confidence Interval 127

References **127**

List of Figures

1 IMS Core Components. 6
2 P-CSCF Discovery 8
3 Subscription Information in HSS. 10
4 IP Multimedia Private Identity in HSS. 11
5 IP Multimedia Public Identity in HSS. 12
6 SDP example. 13
7 IMS Protocol Interfaces. 16
8 Unauthorized Registration Attempt. 18
9 IPsec Association and Authorized Attempt. 19
10 Single Service Center. 20
11 Single Service Center. 21
12 Queue Model for One IMS Component. 24
13 Queueing Network Model for IMS System. 25
14 The Testbed Architecture as A Queueing System Network with Feedback. [1] 29
15 Measured Response Time of SIP 200 and ACK at S-CSCF. [1] 30
16 Response Time of SIP 200 and ACK at S-CSCF Drawn from Derived Functions. 30
17 Message Classes Functionality of First Attempt Registration. 33
18 Open Source IMS Core Playground.[2] 41
19 Alice Profile Configuration. 43
20 myMonster Messaging Test. 44
21 myMONSTER Call Test. 44
22 IMS Bench SIPp Components.[3] 46
23 IMS Bench SIPp Example. 48
24 SIPp Registration Procedure. 49
25 Response Time for IMS Test-bed with 30cps. 50
26 State Flow for Registration Procedure. 52
27 Request flow with associated ports. 54

28 Serving Time at P-CSCF for Registration Procedure. 55
29 Serving Time at I-CSCF for Registration Procedure. 57
30 Serving Time at S-CSCF for Registration Procedure. 57
31 Serving Time at FHoSS for Registration Procedure. 58
32 Serving Time of MySQL for Registration Procedure. 58
33 Call Setup Flow. 60
34 Call Termination Flow. 62
35 Transition Probability. 67
36 Saturation Points of P-CSCF and S-CSCF in All-in-One Machine. 69
37 Saturation Points of I-CSCF and FHoSS in All-in-One Machine. 70
38 Saturation Points of P-CSCF and S-CSCF in 4 Machines. 71
39 Saturation Points of I-CSCF and FHoSS in 4 Machines. 72
40 Unstable System When The Call Rate Is Over The Saturation Point. 73
41 CPU Utilization Curves. 75
42 Start IMS Components With Scripts. 80
43 Call Setup Procedure. 81
44 Call Termination Procedure. 82
45 P-CSCF Module of Open Source IMS System.[4] 93
46 I-CSCF Module of Open Source IMS System.[4] 93
47 S-CSCF Module of Open Source IMS System.[4] 94
48 FHoSS Module of Open Source IMS System.[4] 94
49 FHoSS Management Console. 97
50 FHoSS Database Structure. 101
51 I-CSCF Database Structure. 102
52 myMONSTER Login Screen. 103
53 Kernel Timer Frequency Adjustment. 105

List of Tables

1 3GPP Releases with IMS Features. 5
2 SDP Description Type. [5] 14
3 SIP Request Methods. [6] 15
4 Queue Notations 22
5 IMS Request Classifications. 32
6 Number of Requests in Each IMS Node. 38
7 Call Setup Procedure Steps. 38
8 Average Serving Time (ms) for Registration Procedure. 56
9 Confidence Interval for P-CSCF Serving Time at 98% Confidence. 56
10 Comparison between Measured and Calculated Response time (ms). 59
11 Call Setup Message Type. 61
12 Average Request Serving Time (ms) for 3-procedure Test Case. 63
13 The Calculated Time (ms) for Each Procedure at IMS Nodes. 64
14 Comparison between Measured and Calculated time (ms) for 3 Procedures. 65
15 Service Time of Four IMS Components in All-in-One Machine. 68
16 Service Time of Four IMS Components in 4 Machines. 68
17 CPU Utilization of 4 IMS Components. 73

List of Acronyms

3GPP	Third Generation Partnership Project
AAA	Authentication, Authorization and Accounting
AS	Application Server
CDR	Charging Data Records
CPS	Calls Per Second
CSCF	Call Session Control Function
E-CSCF	Emergency CSCF
FCFS	First Come First Serve
FHoSS	FOKUS Home Subscriber Server
FIFO	First In First Out
FQDN	Fully Qualified Domain Name
HLR	Home Location Register
HSS	Home Subscriber Server
I-CSCF	Interrogating CSCF
IBCF	Interconnection Border Control Function
IETF	Internet Engineering Task Force
iFC	Initial Filter Criteria
IHS	Inadequately Handled Scenarios
IMPI	IP Multimedia Private Identity

IMPU	IP Multimedia Public Identity
IMS	IP Multimedia Subsystem
IPsec	IP Security
LCFS	Last Come First Server
LIFO	First in First Out
MAA	Multimedia Authentication Answer
MONSTER	Multimedia Open InterNet Services and Telecommunication Environment
NAI	Network Access Identifier
NAT	Network Address Translation
NGN	Next Generation Network
P-CSCF	Proxy CSCF
PCRF	Policy and Charging Rules Function
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RTCP	RTP Control Protocol
RTP	Realtime Transport Protocol
SAA	Server Assignment Answer
S-CSCF	Serving CSCF
SCP	Secured CoPy
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SUT	System Under Test
THIG	Topology Hiding Internetworking Gateway

UAC	User Agent Client
UAS	User Agent Server
UE	User Equipment
VLR	Visitor Location Register
VM	Virtual Machine
XML	Extensible Markup Language

Chapter 1

Introduction

The IP Multimedia Subsystem (IMS) is a collaborative work between the 3rd Generation Partnership Project (3GPP) and the Internet Engineering Task Force (IETF) to provide multimedia services over various access technologies. Nowadays it becomes a trend in most mobile operators. They deploy IMS along with the Long Term Evolution (LTE) network to provide subscribers with feature-rich services especially in voice, video and data. At the time of writing, there are more than 70 countries on six continents, of which more than 140 IMS networks have already gone live commercially [7]. This adds up to hundreds of millions of users around the world. The number of subscribers will grow substantially in the near future as the smart phone market gets saturated and new application services are introduced into the IMS. Besides having a robust network, we must evaluate the system performance and foresee future situations in order to serve this number of customers.

There are many methods used to evaluate IMS performance. We can group them into two classes: theoretical evaluation and experimental evaluation. Theoretical evaluation introduces methodology models to get the mathematical results while experimental evaluation uses equipment to measure real-time traffic and then deduces the current system performance. The most recent method [1] combines a queueing model with measurements to produce system performance evaluations. However, the result in this paper is an estimation that only considers two IMS core nodes and neglects the outside impacts on IMS system. Furthermore, this model is specific for the author's test-bed. It is difficult to apply to other complicated IMS networks to achieve reasonable results.

European Telecommunications Standards Institute (ETSI) developed Technical Standard 186-008 for IMS performance evaluation. It provides a framework to test the performance of IMS by collecting success/fail rate, response time and round-trip delays. In this thesis, we propose a new model to evaluate IMS system performance focusing on the response time. Our model considers the IMS network as a queueing network where each IMS node acts as a queue with a processing server. The incoming messages are considered as requests and are classified into different classes. They enter the IMS system by arriving at the queue of the Proxy Call Session Control Function (P-CSCF). After being processed, they are forwarded to other queues. The same process continues until the requests exit the IMS system. Our model will consider all IMS core nodes for the evaluation. In addition, when the IMS system expands, which is the case of a production network, our model can also accommodate the changes. We then build a test-bed system to verify our proposed model. It is built on the Open Source IMS Core System developed by Fraunhofer Institute FOKUS. In order to simulate the procedure, we use IMS Bench SIPp tool to generate the traffic. This is a SIP client running user-created test cases (XML files) to produce mass subscribers' traffic. We verify our model with the IMS registration, call setup and termination procedures. The average response times of requests passing through the queues are measured and fed into our equation to obtain the overall IMS response time.

The thesis is structured as:

- **Chapter 2 - Background.**
  This chapter will provide the basic concepts to understand our proposed model and analysis. The IMS system consists of many different components but we only review four of them in our project. They are three Call Session Control Function (P-CSCF, I-CSCF and S-CSCF) and Home Subscriber Server (HSS). They are the main components that we will build for our test-bed. We also present the protocols (SIP and Diameter) used for these components. Some basic concepts of queues and queueing networks are introduced for our methodology model.

- **Chapter 3 - Analysis.**
  In this chapter, we review previous work relating to IMS performance evaluation. We then point out their drawbacks. Finally we introduce a new queueing model to overcome these problems.

- **Chapter 4 - Testing and Evaluating**
  We build our IMS test-bed system to verify the proposed model. Some basic functional

tests are done. We then use the IMS Bench SIPp to generate traffic for our system. The performance results are collected and verified against our model.

- **Chapter 5 - Conclusion and Future Work**
  The chapter will review and give conclusion for our work. We then introduce some possible future work.

Chapter 2

Background

In this chapter, we review the architecture and the underlying technologies of the IP Multimedia Subsystem. These are the basic concepts and architectures that we are going to use in our analysis and evaluation.

2.1 IP Multimedia Subsystem (IMS) Architecture

The Internet and Cellular world have been enormously evolving for many years. They provide lots of useful services such as messaging, voice, video and data transportation on various medium. The access technologies can be either wire or wireless. The transport technologies vary between the Internet and the Cellular network. The Next Generation Network (NGN) was introduced as an important change to the telecommunication network. The idea of NGN is to use one network to transport all the services by encapsulating data into packets. The traditional circuit-switched networks are replaced by packet-based networks. Besides, NGN also offers the advantages in Quality of Service (QoS), multimedia charging and many other integrated services. The NGN architecture contains many different components but IMS plays a significant role in the core of NGN.

IMS standards have been developed by 3GPP since 1999. The first specification was introduced in 3GPP Release 5. This release provides a general description for IMS components along with QoS specifications which is one of the key IMS features inheriting from the IP-based network. The QoS can be defined based on many factors such as the state of network or the maximum bandwidth allocation for subscribers. Release 6 and 7 define the IMS interworking between the wire and wireless network. Interworking with the old circuit-switch network such as Public Switched Telephone Network (PSTN) is also mentioned. The following releases (starting from Release 8) focus on functionality enhancements for IMS such as service continuity, restoration procedure, interoperability and roaming for voice. In the latest release (Release 12), the extended securities in media plane and peer-to-peer services are concerned. All of these IMS standard developments can be tracked on the 3GPP website [8]. The summary of 3GPP Releases with IMS features is shown in Table 1.

IMS defines three functional layers for the system. They are Transport, Session Control and Application layer. Each layer consists of multiple components with their standard specifications. These collections of components are linked by the standardized interfaces as shown in Figure 1. Being divided into three different functional layers, IMS can achieve efficiency and integrity in providing customer services.

- **Transport layer** provides subscribers with the connectivity to the upper layers of IMS. The subscribers can access IMS using wire (e.g., cable, Ethernet) or wireless (e.g., 3G/LTE, WiMAX, WLAN) technologies. Other legacy systems such as PSTN and non-IMS compatible systems have to go through gateways to reach IMS. This layer has a wide range of network devices and user equipment (UE).
- **Session Control layer** is the core of the IMS system. This layer provides the

Release	***Year***	***Features***
Rel-5	2002 Q1	- First release of IMS.
Rel-6	2004 Q4	- Interoperability of IMS. - Security enhancements.
Rel-7	2007 Q4	- Conferencing support. - Multimedia Telephony Service.
Rel-8	2008 Q4	- Centralize Service control. - Service continuity and restoration procedure. - Application Server descriptions.
Rel-9	2009 Q4	- Emergency call. - Enhancements for service restoration and protection.
Rel-10	2011 Q1	- Enhancements for interoperability.
Rel-11	2012 Q3	- Roaming for voice. - Overload control.
Rel-12	2014 Q2	- Peer-to-Peer Content Distribution Services. - Extended security in media plane.

Table 1. 3GPP Releases with IMS Features.

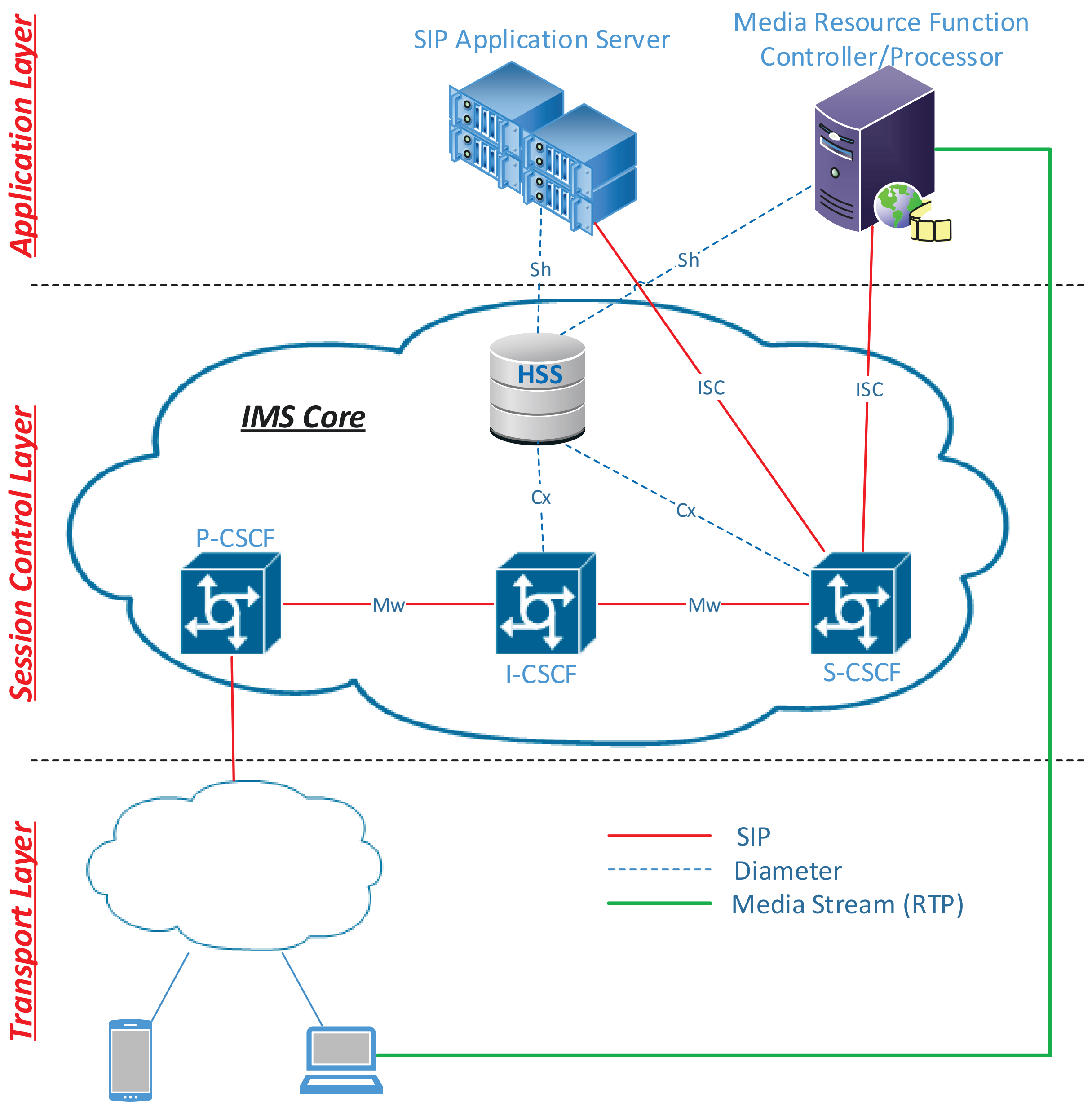


Figure 1. IMS Core Components.

control of communication sessions for subscribers. The Session Initiation Protocol (SIP) is used to establish, maintain and terminate the sessions. In addition, the Diameter protocol is used to retrieve subscription information for Authentication, Authorization and Accounting (AAA). The main components in this layer are the Call Session Control Function (CSCF) and Home Subscriber Server (HSS), which will be described in Section 2.1.1 and 2.1.2. The media gateways connecting to PSTN and other networks are located in this layer.

- **Application layer** is where all the services reside. This layer includes the SIP Application Server (AS), Media Server, etc. It performs many functions such as security, billing, conference call and messaging. These functionalities can be provisioned on centralised or distributed ASs.

2.1.1 Call Session Control Function (CSCF)

The CSCFs reside in the Session Control Layer which is responsible for signalling control between UEs and IMS services, allocating application servers, establishing emergency connection and controlling communication with other networks. They play an important role in the IMS core network. They are a collection of functionalities acting as the entry for every service. The IMS system defines logical functions that do not necessarily have one-on-one corresponding physical components [9]. This means that one function can be implemented on more physical devices or many functions can be mapped on one physical device. Thus, CSCF is normally separated into three logical groups of functions corresponding to three physical devices. They are Proxy, Serving and Interrogating CSCF.

Proxy CSCF (P-CSCF)

The P-CSCF acts as the proxy between the subscribers and the IMS services. All the SIP messages coming from UEs or the IMS network pass through P-CSCF. The UE first performs the P-CSCF discovery process. Then it attaches to a P-CSCF (either at home or visiting network) to register and query for the services. This procedure is shown in Figure 2.

The P-CSCF discovery process can be summarized into three steps:

- **Step 1**: The UE first establishes connectivity with the network. Then it sends a DHCP query asking for the IP address of P-CSCF.

- **Step 2**: After receiving DHCP reply, the UE sends DNS query for the FQDN of P-CSCF.

- **Step 3**: When receiving DNS reply, the UE has the IP address of P-CSCF. It can start the 2-way registration procedure (REGISTER-401-REGISTER-200). The details are explained in Section 2.3.

Besides acting as a proxy for the IMS network, P-CSCF also has other functionalities such as:

- *Establishing IPsec Security Association with UEs*: The registration procedure has two phases. The UE first sends a Register SIP message to the P-CSCF. The message is then forwarded to HSS where the subscriber's information is queried. The 401 Unauthorized message including Cipher Key (CK) and Integration Key (IK) is sent back to challenge the UE. When P-CSCF receives the 401 Unauthorized message, it uses the CK and IK to establish a security tunnel to UE. From now on, all the communication transits through this tunnel.

- *Compressing and decompressing SIP messages*: When the UE is a mobile device, the interface to P-CSCF is air interface and it is bandwidth-limited. The P-CSCF implements compression protocol SigComp to decrease the message size and improve the performance.

- *Charging and policy controlling*: When IMS system generates Charging Data Records (CDRs) and delivers them to the billing system, the P-CSCF can act as a forwarder

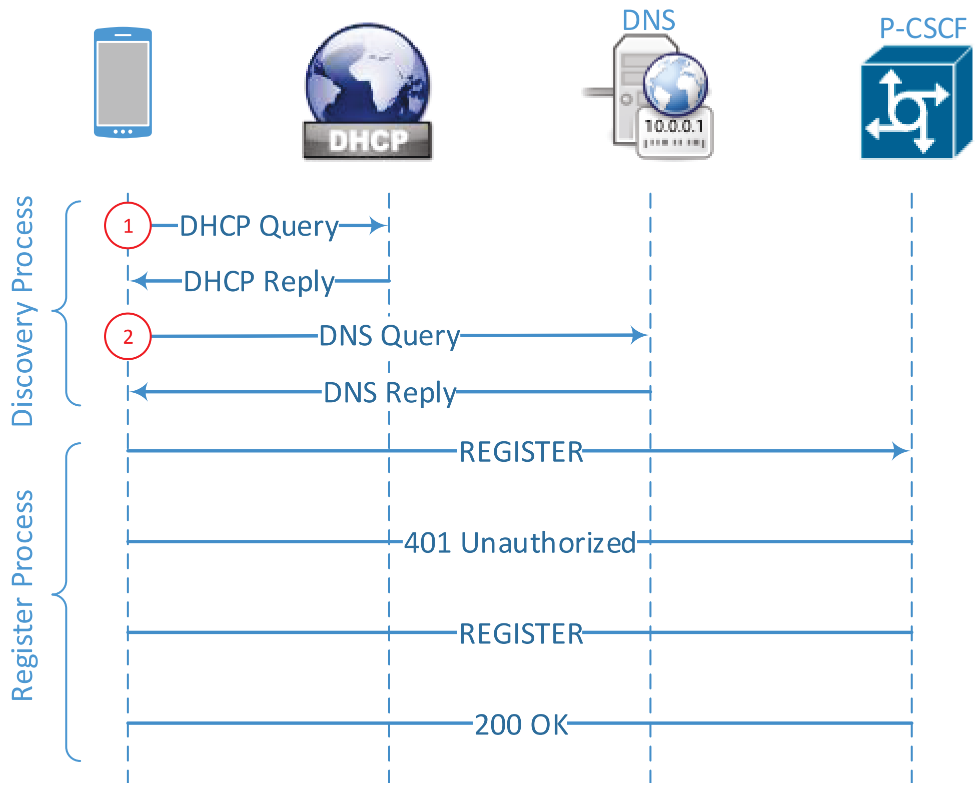


Figure 2. P-CSCF Discovery

for CDRs to the Policy and Charging Rules Function (PCRF). It also can extract information from SIP messages and send them to the PCRF to create policy rules for the call.

- *Identifying emergency calls*: When the P-CSCF receives emergency calls, it forwards them with their locations to the Emergency CSCF (E-CSCF). The E-CSCF will then route them to the 911 Center.

Interrogating CSCF (I-CSCF)

The I-CSCF is located in the home network of the subscribers. The P-CSCF must know the IP address of I-CSCF to forward SIP messages. In order to do that, the P-CSCF extracts the home domain in the SIP Register or the SIP Invite message and consults the DNS server for the IP address results. When the I-CSCF receives the Register or the Invite SIP message, it uses Diameter to contact HSS to find the appropriate S-CSCF. The HSS replies with a list of available S-CSCFs. The I-CSCF then chooses the S-CSCF based on the criteria of the call and the capabilities of the S-CSCF. In the previous 3GPP releases (5 and 6), I-CSCF also has the function of Topology Hiding Internetworking Gateway (THIG). But starting from Release 7, this is moved to Interconnection Border Control Function (IBCF) which performs NAT (Network Address Translation) and firewall functionality.

Serving CSCF (S-CSCF)

The S-CSCF is the main component of an IMS system. It is located at the subscriber's home network. The S-CSCF mainly provides registration service, maintains the user sessions and links the services of Application Servers with UEs. Some other functions of S-CSCF are:

- *Retrieving information from HSS*: User information and subscription data are fetched from HSS during the first registration attempt to challenge a UE. In the second registration attempt the information is compared to the UE‘s info. If they are matched, the registration is successful.

- *Authenticating users*: When S-CSCF asks for subscription information, HSS also replies with Random Number (RAND), Authentication Token (AUTN), Sign Result (XRES), Cipher Key (CK) and Integrity Key (IK). These are used for the user authentication process.

- *Providing media policing*: Another security feature of S-CSCF is inspecting message payload. It can examine to find if unauthorized media types or codecs are used. If the types and codecs don't match to the subscription data of operator's policy, S-CSCF will reject the messages.

- *Maintain timers*: S-CSCF maintains session timer to efficiently detect and release sessions if they are hung.

- *Translate User Identity*: A UE often uses E.164 number as an identity instead of SIP Uniform Resource Identifier (URI). Because the IMS uses only SIP URI for signalling, E.164 must be translated to URI. This can be done by the DNS translation mechanism as defined in RFC2916 [10].

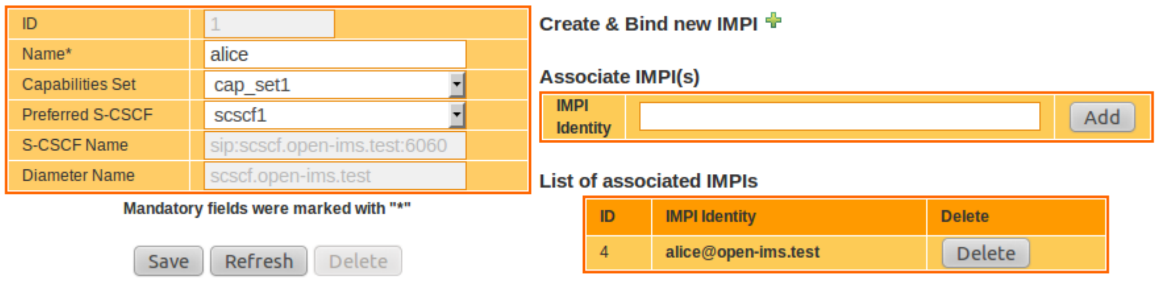
2.1.2 Home Subscriber Server (HSS)

Similar to the Home Location Register (HLR) and Visitor Location Register (VLR) in mobile networks, HSS is a user database containing user profiles, authentication, authorization data and the physical location of users. The components communicating with HSS are Application Servers (ASs) and CSCFs. IMS network can have many HSSs but data of one specific user is stored in one HSS only. An example of subscription data is shown in Figure 3. This is a screenshot taken from our test-bed when Open Source IMS Core system is implemented.

2.1.3 User Identity

In an IMS system, there are various user identities. The two main identities are Private and Public Identity.

- *The private identity* is usually called IP Multimedia Private Identity (IMPI). It is globally unique and is assigned by the operator. It has the format of Network Access Identifier (NAI) defined in RFC 2486 [11]. The IMPI is not used for user communication but for registration, authorization, administration and accounting purposes.



ID	1
Name*	alice
Capabilities Set	cap_set1
Preferred S-CSCF	scscf1
S-CSCF Name	sip:scscf.open-ims.test:6060
Diameter Name	scscf.open-ims.test

Mandatory fields were marked with "*"

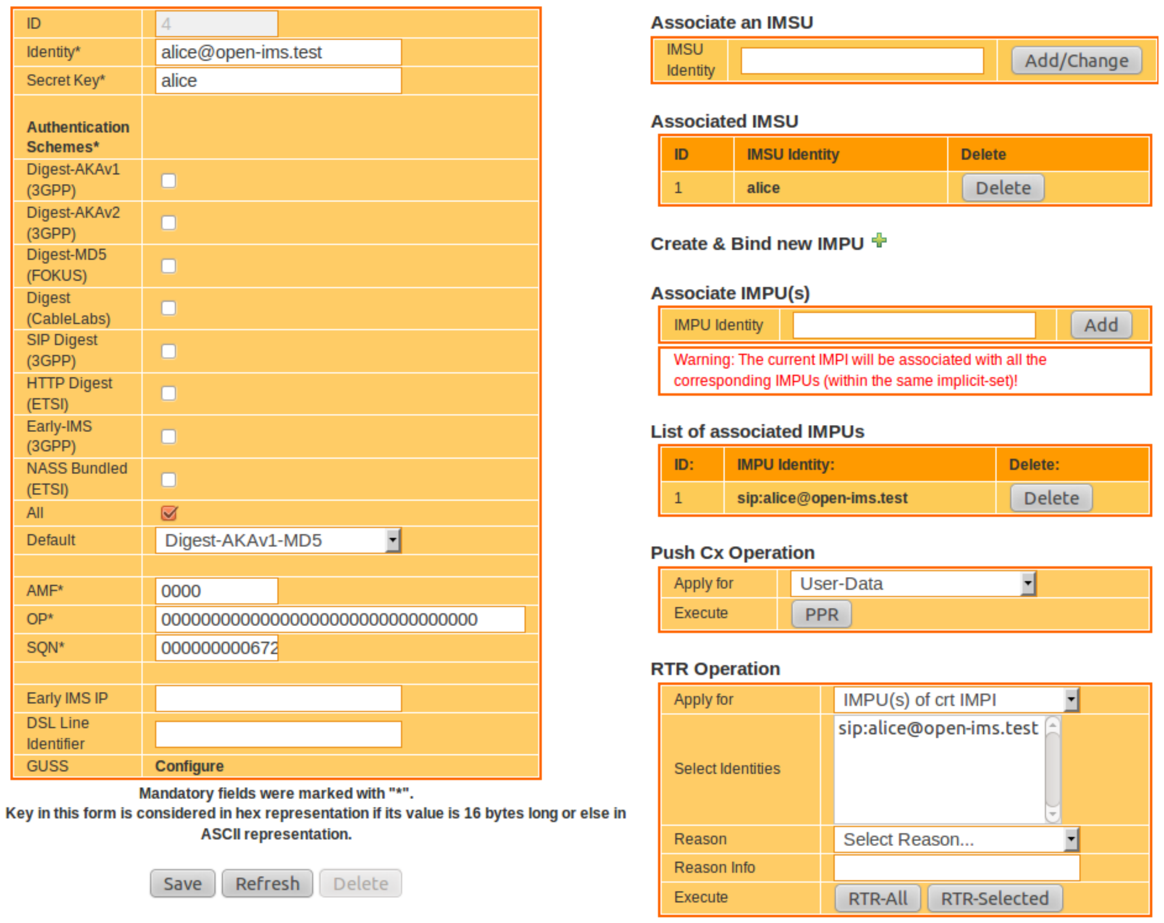
List of associated IMPIs

ID	IMPI Identity	Delete
4	alice@open-ims.test	Delete

Figure 3. Subscription Information in HSS.

Each IMS user has only one IMPI in the form of *user@operator.com*. An example of how IMPI is linked to the Public Identity and authentication schemes is shown in Figure 4.

- *The public identity* is usually called IP Multimedia Public Identity (IMPU). It is an ID that is used for communicating with other users. Multiple IMPUs can be linked to one IMPI. An IMPU can have two alternatives. One of them is SIP URI defined in RFC 3261 and the other is TEL URI defined in RFC 3966. SIP URI is used to communicate inside an IMS network and has the form of *sip:user@operator.com* or *sip:+1-234-567-8900@:operator.com*. TEL URI is used to communicate from IMS to PSTN and has the form of *URITEL:+1-234-567-8900*. An example of how IMPU is linked to the IMPI and its charging info is shown in Figure 5.



ID	4
Identity*	alice@open-ims.test
Secret Key*	alice
Authentication Schemes*	
Digest-AKAv1 (3GPP)	☐
Digest-AKAv2 (3GPP)	☐
Digest-MD5 (FOKUS)	☐
Digest (CableLabs)	☐
SIP Digest (3GPP)	☐
HTTP Digest (ETSI)	☐
Early-IMS (3GPP)	☐
NASS Bundled (ETSI)	☐
All	✓
Default	Digest-AKAv1-MD5
AMF*	0000
OP*	00000000000000000000000000000000
SQN*	000000000672
Early IMS IP	
DSL Line Identifier	
GUSS	**Configure**

Mandatory fields were marked with "*".
Key in this form is considered in hex representation if its value is 16 bytes long or else in ASCII representation.

Associate an IMSU

IMSU Identity		Add/Change

Associated IMSU

ID	IMSU Identity	Delete
1	**alice**	Delete

Create & Bind new IMPU

Associate IMPU(s)

IMPU Identity		Add

Warning: The current IMPI will be associated with all the corresponding IMPUs (within the same implicit-set)!

List of associated IMPUs

ID:	IMPU Identity:	Delete:
1	**sip:alice@open-ims.test**	Delete

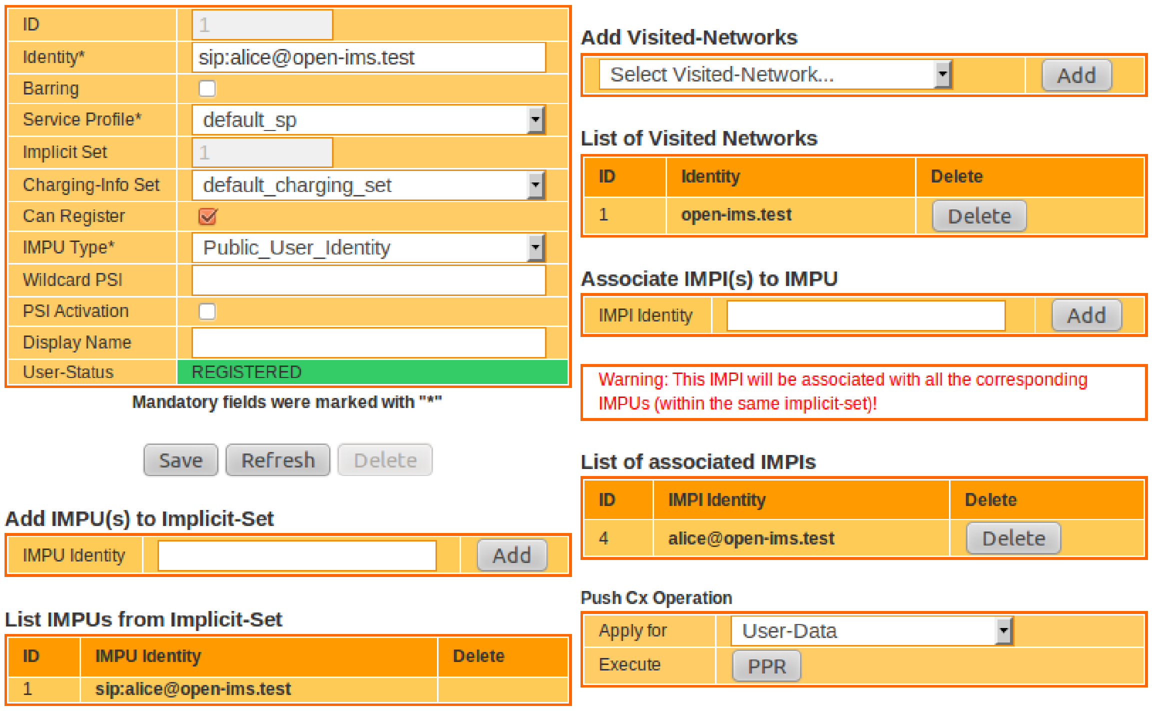
Push Cx Operation

Apply for	User-Data
Execute	PPR

RTR Operation

Apply for	IMPU(s) of crt IMPI
Select Identities	sip:alice@open-ims.test
Reason	Select Reason...
Reason Info	
Execute	RTR-All RTR-Selected

Figure 4. IP Multimedia Private Identity in HSS.



ID	1
Identity*	sip:alice@open-ims.test
Barring	☐
Service Profile*	default_sp
Implicit Set	1
Charging-Info Set	default_charging_set
Can Register	✓
IMPU Type*	Public_User_Identity
Wildcard PSI	
PSI Activation	☐
Display Name	
User-Status	REGISTERED

Mandatory fields were marked with "*"

Add IMPU(s) to Implicit-Set

IMPU Identity

List IMPUs from Implicit-Set

ID	IMPU Identity	Delete
1	sip:alice@open-ims.test	

Add Visited-Networks

Select Visited-Network...

List of Visited Networks

ID	Identity	Delete
1	open-ims.test	Delete

Associate IMPI(s) to IMPU

IMPI Identity

Warning: This IMPI will be associated with all the corresponding IMPUs (within the same implicit-set)!

List of associated IMPIs

ID	IMPI Identity	Delete
4	alice@open-ims.test	Delete

Push Cx Operation

Apply for	User-Data
Execute	PPR

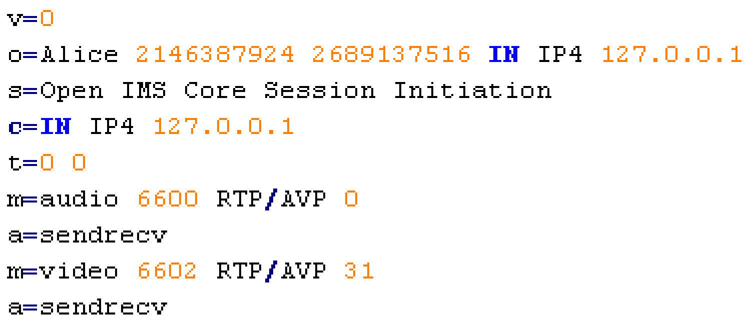
Figure 5. IP Multimedia Public Identity in HSS.

2.2 Protocols in IMS

There are many protocols being used for IMS system such as Session Initiation Protocol (SIP), Diameter, SigComp, Realtime Transport Protocol (RTP), RTP Control Protocol (RTCP) and IP Security (IPsec). In this section we present the SIP and Diameter protocols which are used the most in our test-bed for evaluation.

2.2.1 Session Initiation Protocol (SIP)

From the UE's point of view, SIP is an important protocol used to create, maintain and terminate user session. It is independent of TCP, UDP and IPv4/6. SIP is a text-based protocol for a client/server model. The protocol is developed and defined by IETF under RFC 3261. It was initially developed for session control in media services. Skype and Facetime are two well-known examples of its use. When SIP was adopted to the ISM system, it became so widely known that SIP was thought to be specific for only IMS. Three main SIP concepts are Session Description Protocol (SDP), SIP Request and SIP Response.



```
v=0
o=Alice 2146387924 2689137516 IN IP4 127.0.0.1
s=Open IMS Core Session Initiation
c=IN IP4 127.0.0.1
t=0 0
m=audio 6600 RTP/AVP 0
a=sendrecv
m=video 6602 RTP/AVP 31
a=sendrecv
```

Figure 6. SDP example.

Session Description Protocol

As the name speaks for itself, SDP describes a session between users. It is defined in RFC 2327 and RFC 3264. Minimum information of SDP includes session name, active time of session and network information in order to establish a session. SDP description types are shown in Table 2. Figure 6 shows an example of SDP. This is a functional test in our test-bed demonstrating the call procedure between users Alice and Bob. SDP contains the IP address of Alice 127.0.0.1 (because we built IMS system on the loopback interface, all the components have the same IP but different port numbers), she wants to have an audio call on port 6600 and a video call on port 6602. The codec used for audio is G711 (m=0) and for video is H261 (m=31).

SIP Request

A SIP Request is used to perform a SIP action that is included in the request start line of the message. It has the form: "*Method + Request-URI + SIP-Version*". The *Request-URI* is the user to whom the request is sent. The *SIP-version* can be either "SIP" or "SIP/2.0". The *Method* specifies what the request wants to do. Some examples of Method are shown in Table 3.

SIP Response

A SIP Response is the reply of the SIP server to a SIP Request. It is distinguished by the start line of the message. It has the form: "SIP-Version + Status-Code + Reason-Phrase". The *Status-Code* is a 3-digit number indicating the response type of certain requests. The *Reason-Phase* is a short description for Status-Code. There are six categories of response [6]:

- 1xx: Provisional – The request has been received and the server continues to process the request.

- 2xx: Success – The action was successfully received, understood and accepted.
- 3xx: Redirection – Further action needs to be taken in order to complete the request.
- 4xx: Client Error – The request contains bad syntax or cannot be fulfilled at this server.
- 5xx: Server Error – The server failed to fulfill an apparently valid request.

Type	***Description***
	Session Description
v	protocol version
o	owner/creator and session identifier
s	session name
i	session information
u	URI of description
e	email address
p	phone number
c	connection information - not required if included in all media
b	bandwidth information
z	time zone adjustments
k	encryption key
a	zero or more session attribute lines
	Time Description
t	time the session is active
r	zero or more repeat times
	Media Description
m	media name and transport address
i	media title
c	connection information - optional if included at session-level
b	bandwidth information
k	encryption key
a	zero or more media attribute lines

Table 2. SDP Description Type. [5]

Method	***Description***
INVITE	The user is invited to a call session.
ACK	Confirmation when INVITE message is received.
BYE	Terminate a call session.
CANCEL	Terminate pending requests.
OPTIONS	Asking for server capabilities.
REGISTER	Register the client to the system.

Table 3. SIP Request Methods. [6]

- 6xx: Global Failure – The request cannot be fulfilled at any server.

2.2.2 Diameter Protocol

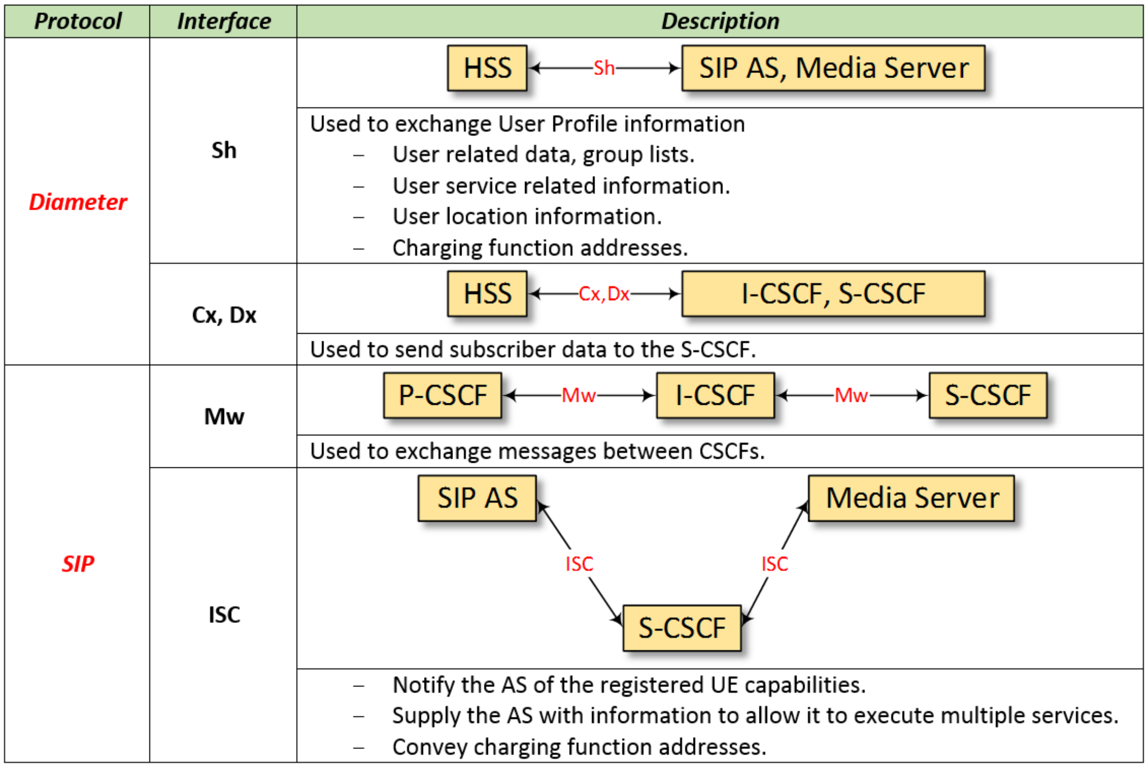
The Diameter protocol is an advanced evolution of Authentication, Authorization and Accounting (AAA) protocols such as RADIUS, TACACS, Kerberos and COPS. RADIUS was originally designed for Point-to-Point Protocol (PPP) networks. It lacks scalability, and thus it cannot be deployed in large networks. Furthermore, it does not support roaming which is the main requirement of mobile networks. RADIUS and TACACS have their own advantages but they are not suitable for today's networks. Diameter was developed to fulfill these requirements. In addition, Diameter supports Extensible Authentication Protocol (EAP) which is commonly used as a security mechanism to prevent fraudulent authentication. Diameter also offers credit control, identity validation, access policy and charging function for clients on the network [12]. A summary of Diameter and SIP interfaces between IMS components is shown in Figure 7.

2.3 Registration Procedure in IMS

We will present the registration procedure in this section. The general registration procedure has three phases: Unauthorized Registration, IPsec Association and Authorization. These phases involve three CSCF types and HSS. They use SIP and Diameter through the protocol interfaces as shown in Figure 7.

The ***Unauthorized Registration Attempt*** is shown in Figure 8 with the following steps:

(1) UE sends a REGISTER message to P-CSCF. It is sent on the standard SIP port



Protocol	Interface	Description
Diameter	Sh	HSS ←Sh→ SIP AS, Media Server
		Used to exchange User Profile information – User related data, group lists. – User service related information. – User location information. – Charging function addresses.
	Cx, Dx	HSS ←Cx,Dx→ I-CSCF, S-CSCF
		Used to send subscriber data to the S-CSCF.
SIP	Mw	P-CSCF ←Mw→ I-CSCF ←Mw→ S-CSCF
		Used to exchange messages between CSCFs.
	ISC	SIP AS ←ISC→ S-CSCF ←ISC→ Media Server
		– Notify the AS of the registered UE capabilities. – Supply the AS with information to allow it to execute multiple services. – Convey charging function addresses.

Figure 7. IMS Protocol Interfaces.

5060.

② ③ P-CSCF receives the REGISTER message. Then it translates the Home Network Domain to an IP address.

④ The REGISTER message is forwarded to the IP Address of I-CSCF (resolved from the DNS query of step 3).

⑤ I-CSCF queries HSS for the S-CSCF.

⑥ HSS replies with the names and capabilities of S-CSCF.

⑦ I-CSCF selects the S-CSCF based on the S-CSCF capabilities.

⑧ I-CSCF forwards the REGISTER message to the S-CSCF.

⑨ S-CSCF queries HSS for the Authentication of this UE.

⑩ HSS replies with Random Number (RAND), Authentication Token (AUTN), Sign Result (XRES), Cipher Key (CK) and Integrity Key (IK.)

(11) The UE is currently unauthorized. S-CSCF sends 401 Unauthorized message to I-CSCF with RAND, AUTN, CK and IK.

(12) I-CSCF forwards 401 Unauthorized message to P-CSCF with RAND, AUTN, CK and IK.

(13) P-CSCF saves CK and IK which are used to established IPsec tunnel to UE.

(14) P-CSCF selects connection ports for both P-CSCF and UE.

(15) P-CSCF forwards 401 Unauthorized to UE with the allocated ports, RAND and AUTN (removed CK, IK).

(16) UE verifies to the AUTN and computes RES which will be sent back to HSS for user authentication.

Figure 9 shows the ***IPsec Security Association Establishment*** and the ***Authorized Registration Attempt*** with the following steps:

(17) (18) (19) (20) IPsec tunnel is established between UE and P-CSCF using the allocated port, CK and IK.

(21) UE now sends a REGISTER message (including the computing RES value) over IPsec tunnel to P-CSCF.

(22) P-CSCF forwards the RESGISTER message to I-CSCF.

(23) I-CSCF queries HSS for the S-CSCF.

(24) HSS replies with the S-CSCF names and capabilities.

(25) I-CSCF selects the S-CSCF based on the S-CSCF capabilities.

(26) I-CSCF forwards the REGISTER message to S-CSCF.

(27) S-CSCF requests UE subscription information from HSS.

(28) HSS replies with the subscription info.

(29) S-CSCF compares the RES value with the XRES value (computed in step 10.)

(30) Assuming the values match, P-CSCF sends a 200 OK message to I-CSCF.

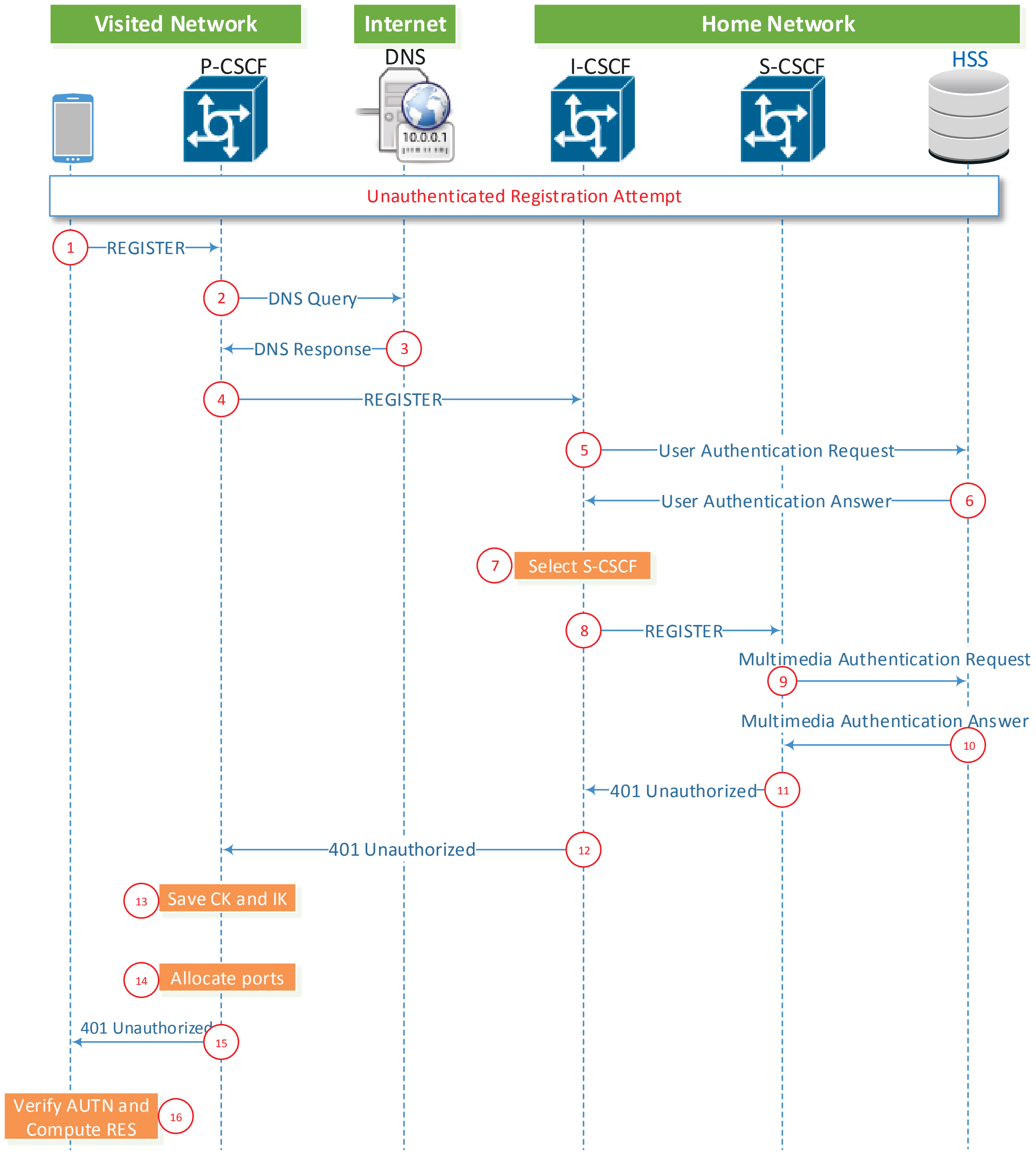


Figure 8. Unauthorized Registration Attempt.

(31) I-CSCF forwards the 200 OK message to P-CSCF.

(32) P-CSCF forwards the 200 OK message to UE. The registration is completed.

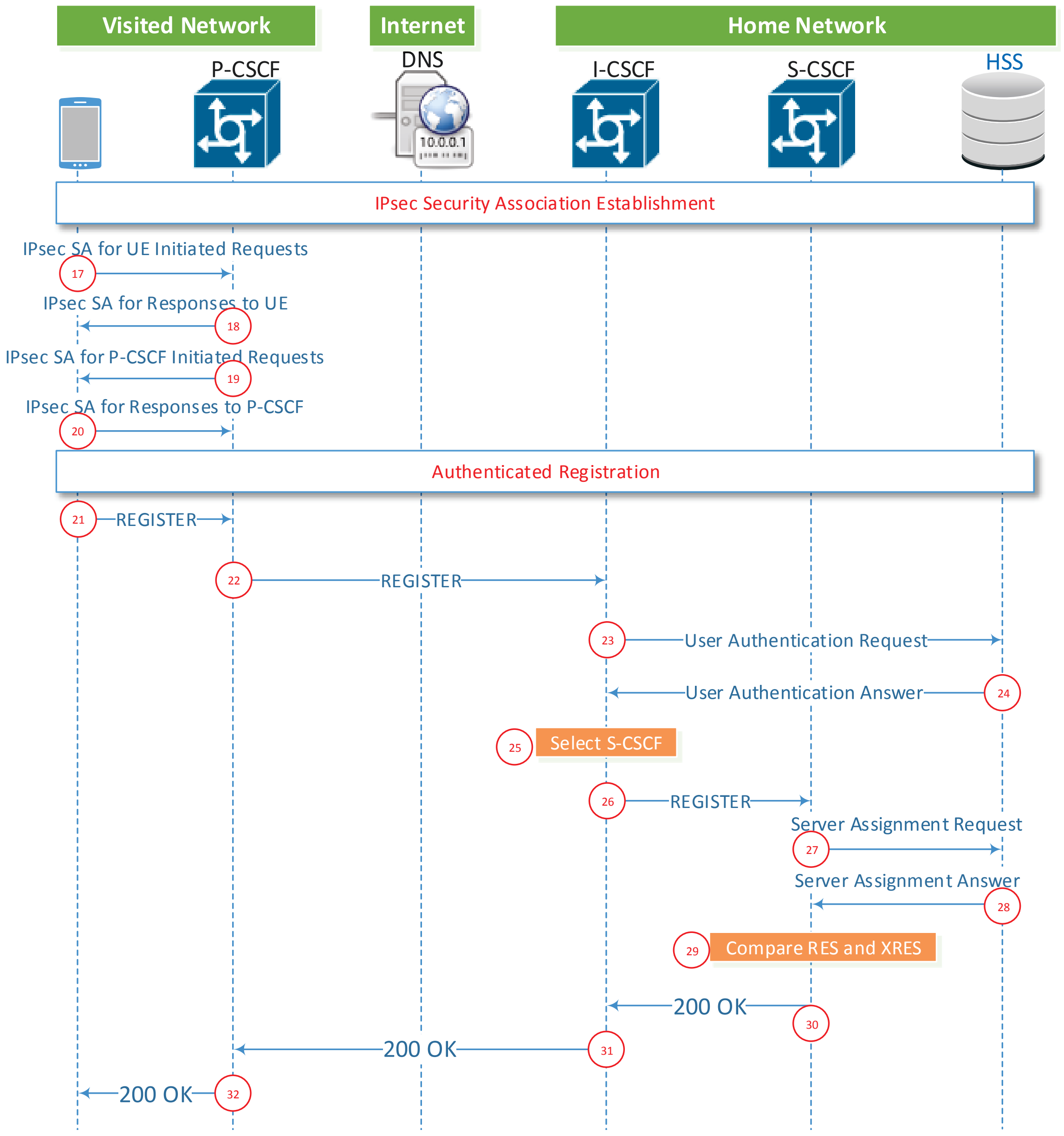

Visited Network
Internet
Home Network
P-CSCF
DNS
10.0.0.1
I-CSCF
S-CSCF
HSS
IPsec Security Association Establishment
IPsec SA for UE Initiated Requests
17
IPsec SA for Responses to UE
18
IPsec SA for P-CSCF Initiated Requests
19
IPsec SA for Responses to P-CSCF
20
Authenticated Registration
21
REGISTER
22
REGISTER
23
User Authentication Request
User Authentication Answer
24
25
Select S-CSCF
26
REGISTER
Server Assignment Request
27
Server Assignment Answer
28
29
Compare RES and XRES
200 OK
30
200 OK
31
200 OK
32

Figure 9. IPsec Association and Authorized Attempt.

2.4 Queueing Network

Because the way IMS components interact with each other is random, we will use stochastic models to evaluate IMS performance and probability theory to analyze its unpredictable events. Specifically the system performance analysis will be conducted using queueing theory to study the interaction between CSCFs and HSS.

2.4.1 Queue Definition

Each IMS component is considered to be a single queue (or service center) with a single processing server. The incoming messages are called customers or requests. A typical single service center is shown in Figure 10. A request has to be processed by the server before leaving the service center. If the server is busy, it must wait in the buffer until the service is available. A queue (or service center) is described by:

- **Arrival Pattern**. The service ability of a system first depends on the average request arrival rate λ. If the arrival rate is greater than the service rate, the requests have to stay in the queue and wait for their turn. The pattern of how requests arrive (distribution function) is also important. It tells how the traffic is distributed during a specific time interval. The three common types of distribution for arrival rate are Exponential, Deterministic and General distribution. In our analysis we will consider our queues with exponential arrival rate.
- **Service Pattern**. This factor has the same characteristics as Arrival Pattern (service time and distribution). We will use exponential distribution with the average duration of $^1/_\mu$ where μ is the service rate.
- **Numbers of Servers**. A queue can have more than one processing server. Each server can serve the requests individually and concurrently. There are three types of server: single, multiple and infinite. In a single server, the requests are served one by one. The incoming requests must wait for the previous one to be finished before

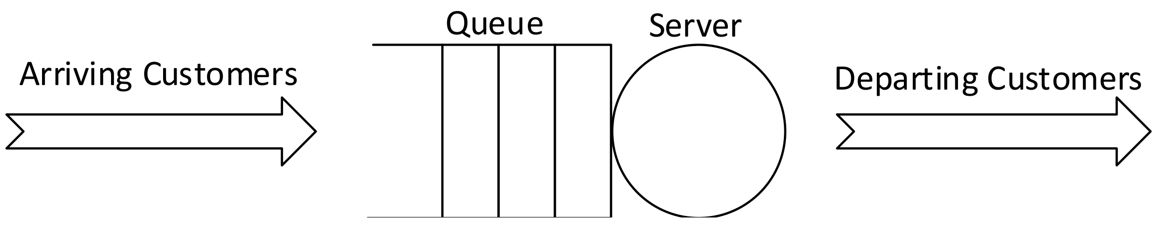


Figure 10. Single Service Center.

entering the server. In a multiple server queue, there is a fixed number of servers that can process the requests concurrently. In an infinite server, there are as many servers as the incoming requests. As soon as the requests arrive, they are served. There are no queues in this case.

- **Queue Capacity**. In the case of single server or multiple servers, there are situation when requests cannot be processed and have to wait in the queue for their turns. The number of requests grows and fills up the queue. The service center might either notify the source to suspend the requests or ignore and drop the requests until the resource is available.

- **Request Population**. This has the same characteristics as the Queue Capacity. The source can have a finite or infinite number of requests. The request population can also be divided into classes. Each class has its own characteristics. Different classes can have different arrival rates or can be served differently in the server.

- **Service Discipline**. This is one of the important features of a service center. It defines how the requests are served when they enter the queue. It can be classified as:

  - *First Come First Serve* (FCFS) or *First In First Out* (FIFO).
  - *Last Come First Serve* (LCFS) or *Last In First Out* (LIFO).
  - *Random Selection for Service* (RSS).
  - *Priority* (PRI). Requests having different priorities are served in different orders.

The above descriptions are expressed in notations that are shown in Figure 11. This is called Kendall's notation. The descriptions are shown in Table 4. The following examples clarify the usage for notations in queueing network:

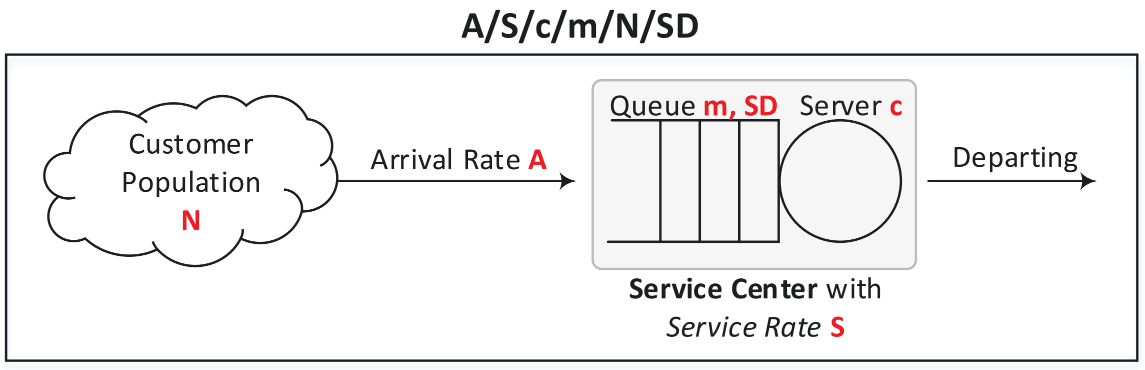


Figure 11. Single Service Center.

A	Arrival Rate.	**M** – Markov (Exponential Distribution).
		G – General Distribution.
		D – Deterministic Distribution.
S	Service Rate.	**M** – Markov (Exponential Distribution).
		G – General Distribution.
		D – Deterministic Distribution.
c	Number of Servers.	
m	Queue Capacity/Length.	Infinite by default.
N	Customer Population.	Infinite by default
SD	Service Discipline.	**FCFS** – First Come First Serve (default)
		LCFS – Last Come First Serve
		RSS – Random Selection for Service
		PRI – Priority

Table 4. Queue Notations

- When we specify a queue as "$M/M/3/6/\infty/FCFS$" it means:
  - **Arrival Rate M** follows Markov Distribution.
  - **Service Rate M** follows Markov Distribution.
  - There are **3 server**s.
  - **Queue length** is **6**.
  - **Infinite** ∞ Customer Population.
  - **Service Distributio**n is **FCFS**.

  ⇒ Fields with default values can be omitted. It is then reduced to: **M/M/3/6**.

- When we specify a queue as "$M/G/1/\infty/\infty/FCFS$" it means:
  - **Arrival Rate M** follows Markov Distribution.
  - **Service Rate G** follows General Distribution.
  - There is **1 server**.
  - **Queue length** is ∞.
  - **Infinite** ∞ Customer Population.
  - **Service Distributio**n is **FCFS**.

  ⇒ Fields with default values can be omitted. It is then reduced to: **M/G/1**.

Besides the arrival rate λ and the service rate μ we also use **traffic intensity** ρ to describe a queue. A system is said to be stable when $\rho < 1$. It is commonly used in the final evaluation formula of a system and is defined as:

$$\rho = \frac{\lambda}{c \times \mu}$$

2.4.2 Queueing Network

Each IMS component will be modelled as a queue (Figure 12) in our analysis with the following definitions:

T	The length of *time* we observe the system.
A	The number of request *arrivals* we observe.
C	The number of request *completions* we observe.
V	The average *number of jobs* in the system.
B	The total amount of time during which the system is *busy*. ($B \leq T$)
U	The percentage of time during which the system is in *use*.
S	*Service Time* - The average time the system serves a job.
W	*Residence Time* - The average time a job spends in the system.
R	*Response Time* - Is equal to the residence time minus the think time.
Z	*Think Time* - The time a job spends before entering the system.
X	*Throughput* - The number of completions during a time length.

This queue follows **Little's law** [13] which states that the average number of requests in a system is equal to the product of the throughput of the system and the average time spent in that system by a request:

$$N = XW$$

In addition, these definitions also have the following relationships:

- $X = \frac{C}{T}$
- $U = \frac{B}{T}$
- $S = \frac{B}{C}$
- $R = W - Z$

We then build a corresponding queueing network to the IMS network as in Figure 13. Each queue has its own parameters as defined. They are connected to each other to represent an IMS system. The requests coming into the system are forwarded to the first

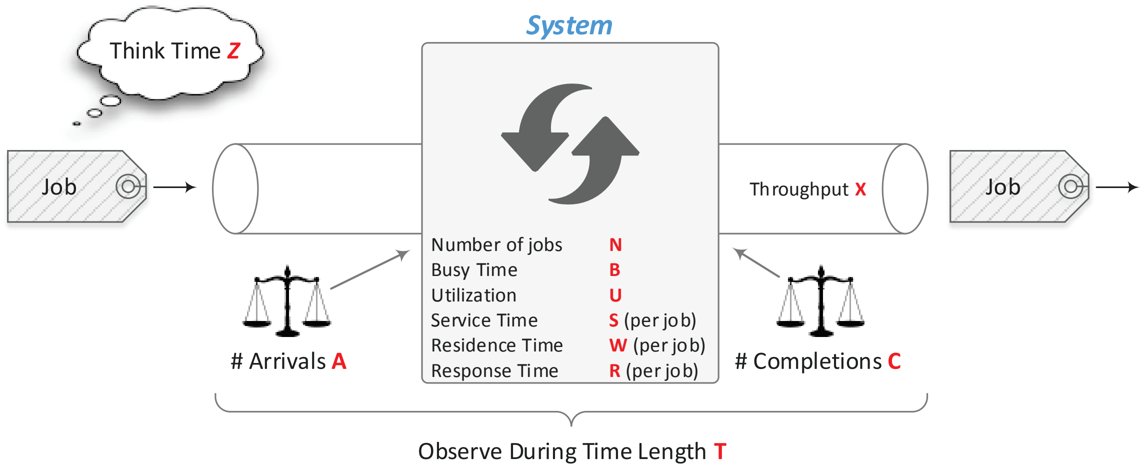


Figure 12. Queue Model for One IMS Component.

queue. They are processed and forwarded to the next queue with a certain probability. The process continues until the jobs are done and the requests exit the system.

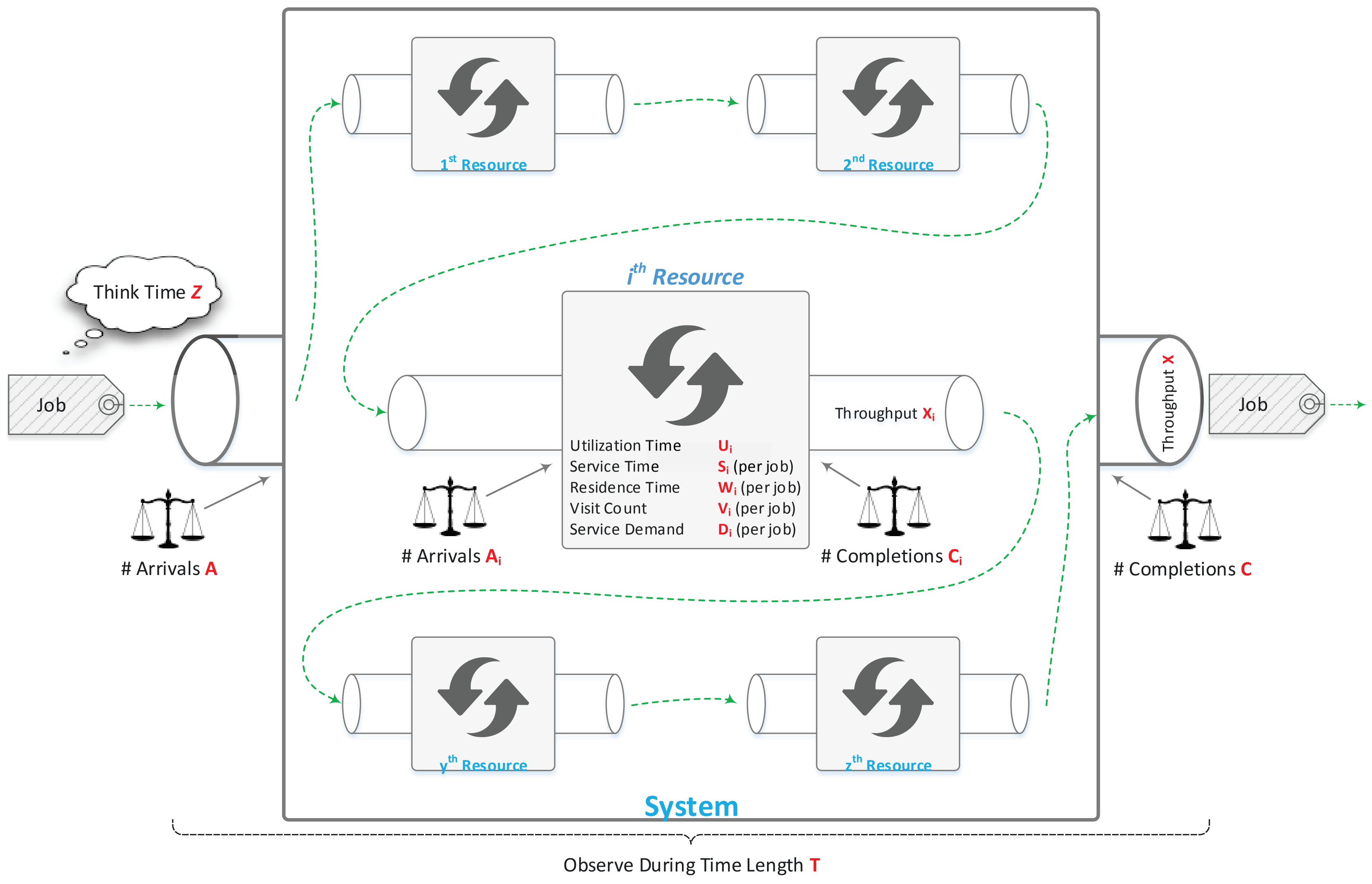


Figure 13. Queueing Network Model for IMS System.

Chapter 3

Analysis

In IMS system the user registration is crucial. An unsuccessful registration will prevent subscribers from accessing the services. A proper analysis and evaluation of registration procedure has to be performed. By doing that, we will have a full understanding of the whole system and have appropriate provision, expansion, restoring and backup plans. In the following section we review previous work and propose our new method to evaluate the registration time in IMS. In addition, our model is also expanded to evaluate the call setup and termination procedure.

3.1 Related Work

The number of IMS underlying technologies is so large that we need to have common ground rules for service evaluation. The ETSI TISPAN group developed IMS/NGN performance benchmark standards for this purpose. It provides a framework to test the performance characteristics of IMS under realistic conditions by collecting success/fail rate, response time and round-trip delays. The specifications are mentioned in ETSI TS 186 008 [14] which includes four parts. The first part explains the descriptions, processes, architectures for the IMS benchmark model. The second part contains the benchmarking scenarios with metrics and design objectives. It also describes the IMS configuration parameters. The third part defines test procedure benchmark, traffic set and traffic-time profile that will be fed into the IMS system. The fourth part specifies the reference load parameters for user-cases in part two. Generally, a benchmark test consists of multiple Test Systems (TSs) and a System Under Test (SUT). The TS generates and injects the traffic into the SUT. According to the specification, TS must be able to execute various scenarios with defined ratios in the traffic profile. It must have the ability to pass information to the SUT or other TSs for synchronization purposes. When the SUT receives the traffic, it reacts accordingly and generates performance statistics. These results are collected during the test and are sent back to TSs after the test to generate a performance report. There are various IMS performance analysis papers that implement and use the ETSI benchmark standards. The queueing methodological approach was used in only a few of them. We will review these papers in the following paragraphs.

In paper [15], the author's purpose is to implement several ETSI benchmarking scenarios and run them on an Open IMS Core system. It is named "workload model". It is composed of three elements: scenarios (including IMS registration, re-registration, de-registration, voice call and messaging), benchmark tests (including the running ratio of each scenarios) and test report (including performance results). This paper presents the first implementation of the ETSI standard. The performance results are shown through the graph of call creation/error rate, CPU, memory consumption and the average response time for the INVITE message though IMS core network.

The papers [16] and [17] also use the same ETSI specifications to evaluate the IMS performance benchmark on the open source IMS system. The authors perform system stress tests using SIPp generator and obtain the results for Inadequately Handled Scenarios (IHS), retransmission rate and CPU utilization. These results are projected on the graph with the call rates and execution time. It is found that most of the issues are related to the configuration of the Open IMS Core components. In addition, the HSS is the failure point for the whole system.

The paper [18] proposes three methods to evaluate the call rate and IMS behaviours by using SIPp, PROTOS and Spectra 2 SE tools. The author first studies the IMS performance by using the SIPp traffic generator. The tool operates with a User Agent Server (UAS) and a User Agent Client (UAC). The UAC sends the traffic through an IMS system to reach the UAS. The purpose of this test is to measure the IMS capacity in term of maximum call rate and corresponding load of the system. The second method which uses the PROTOS tool [19] focuses on the robustness and security of IMS system when it receives malformed INVITE requests. The test suite contains 4527 SIP INVITE tests which can be classified into 54 categories. Some common categories are malformed SIP version, content/type, SIP-URI, SIP-tag. The test fails when the IMS system stops running, hangs up, restarts or consumes all CPU or memory. In the third method, the author uses the Spectra2 SE tool to observe the behaviours and the responses of the IMS system to the standard SIP requests.

In paper [20], the purpose is to study the end-to-end signalling delay between user agents and IMS nodes. The author implements SIP Stack Seagull software as the test-bed and measures the delays. Then he compares them with the performance results from the Network Simulator NS2. The conclusion is that the maximum delay lies at S-CSCF therefore an efficient IMS design must focus on the bottleneck at S-CSCF.

While most papers focus only on the practical aspects by using an IMS test-bed along with various testing tools to collect measurements according to the ETSI benchmark standards, there are only a few papers using queueing theory and measurement results to achieve a better IMS evaluation. In the paper [21], the author presents an IMS model based on an M/M/1 queueing system with feedback having the same IMS entities, signalling and services. He proposes this model for simulation of data and voice services on IMS architecture. It consists of two P-CSCFs, five S-CSCFs, one I-CSCF and one HSS. In this model, no specific mathematical formulas are given to evaluate the performance, only the algorithm for the S-CSCF assignment is emphasized. Three designs of the S-CSCF assignment are evaluated based on parameters such as the number of messages in the queue or the server utilization or the combination of the two parameters. It is shown through the simulation results that the best algorithm is based on the actual number of messages in the queue. The method based on the combination of messages in the queue and server utilization is best for a high system load only.

In a later paper [1], the author focuses on the performance evaluation of response time of signalling through each IMS nodes. Compared to the proposed model in [21] which uses an M/M/1 queue, this paper uses an M/G/1 queue which corresponds more with the real IMS network. The author introduces a test-bed with the queueing network as shown in Figure 14. This consists of one P-CSCF, one S-CSCF, one I-CSCF and one HSS. These four IMS components are implemented using Open IMS core software. The input traffic is defined by the Poisson arrival process and is generated using the IxLoad application. The IMS test-bed is run though a range of load intensities from 25cps to 500cps. The measurement results are graphed for each message type. From the measured data, we can see that the response time is linearly increased and saturated at a certain call rate. An example of measured time of the SIP 200 and SIP ACK message at P-CSCF is shown in Figure 15. The measured time for the SIP 200 and SIP ACK increases linearly from 25cps to 100cps and saturates at around 0.52s for call rate over 100cps. The measured time for the MAA and SAA message also increases linearly from 25cps to 500cps. However, the author defines the response time with the exponential or logarithm functions and derives trend-lines based

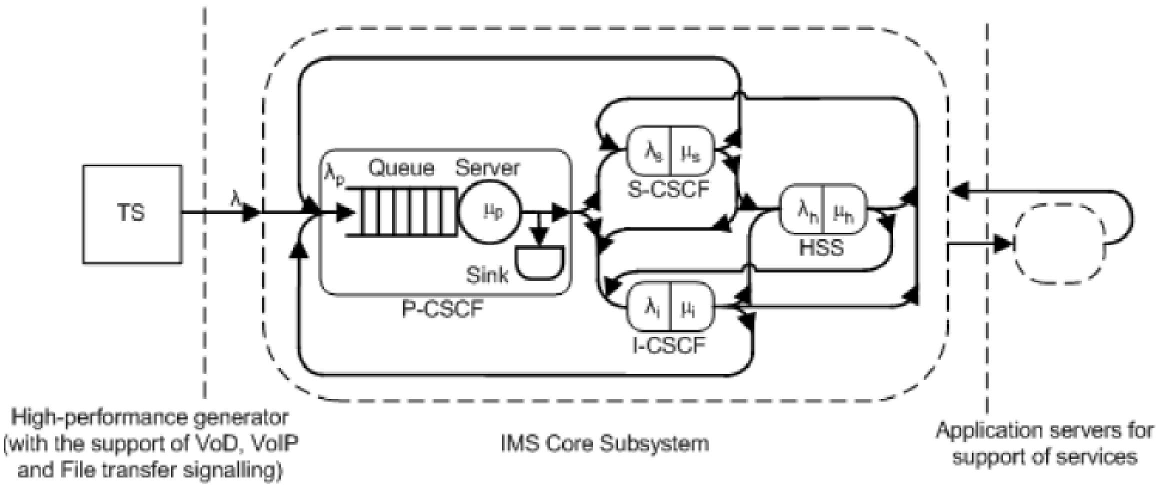


Figure 14. The Testbed Architecture as A Queueing System Network with Feedback. [1]

on these obtained measurements. For example, the SIP 200 for INVITE request has the function of $f(x) = -277.9 \times x^{-1.952} + 0.536$ or the SIP ACK message has the function of $f(x) = -115.9 \times x^{-1.711} - 0.5348$. The parameter x is the load generated by the IxLoad application. These derived functions are graphed in Figure 16. Messages with measured service time less than 1ms are omitted. This evaluation method does not provide general functions to evaluate an extended IMS system where the number of CSCFs can be over a few hundreds. The given functions are derived from the small data set of the author's test-bed. In addition, the IMS components can be combined or separated in the production network. These functions can not evaluate this change. Thus, the model does not deal properly with the IMS scalability issue.

In the delay evaluation, the author neglects the effects of lower signalling and the impact of delays outside the IMS system. The response time for a registration is estimated as [1]:

$$D_{REG} \approx D_{MAA \to 401} + D_{SAA \to 200}$$

$D_{MAA \to 401}$ is the delay from the time when S-CSCF receives the Multimedia Authentication Answer (MAA) message to the time when S-CSCF sends a 401 Unauthorized message in the first registration attempt. $D_{SAA \to 200}$ is the delay from the time when S-CSCF receives a Server Assignment Answer (SAA) message to the time when S-CSCF sends a 401 Unauthorized message in the second registration attempt. As we can see from this equation, the response time depends on only the S-CSCF while neglecting other CSCFs and HSS. This might be only the case in the author's test-bed where less traffic load is generated. However,

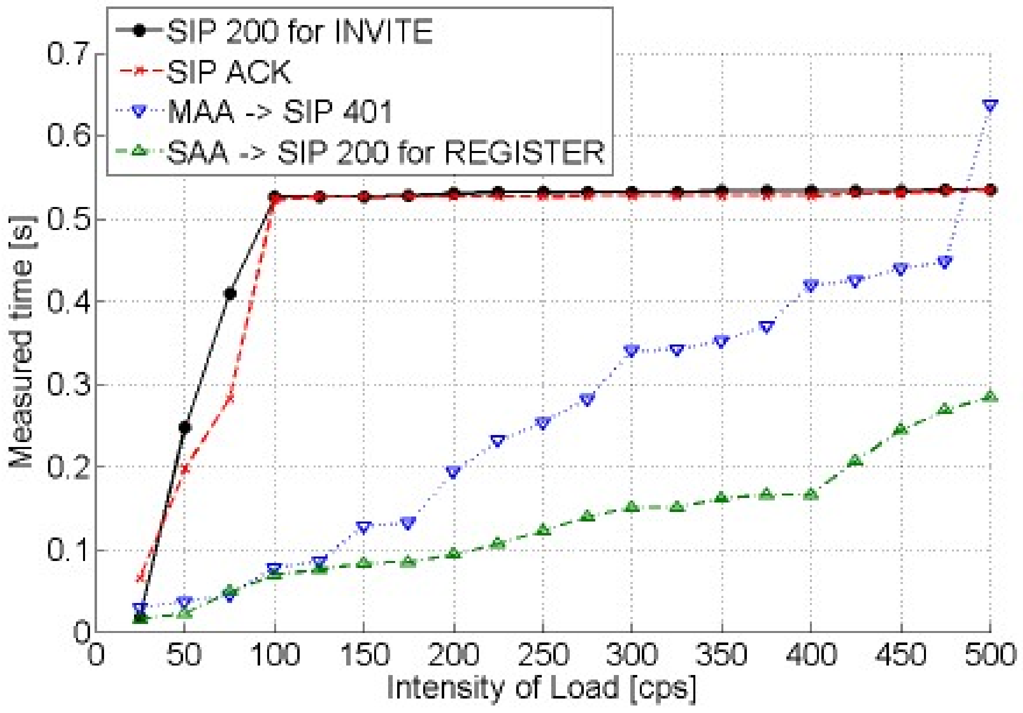


Figure 15. Measured Response Time of SIP 200 and ACK at S-CSCF. [1]

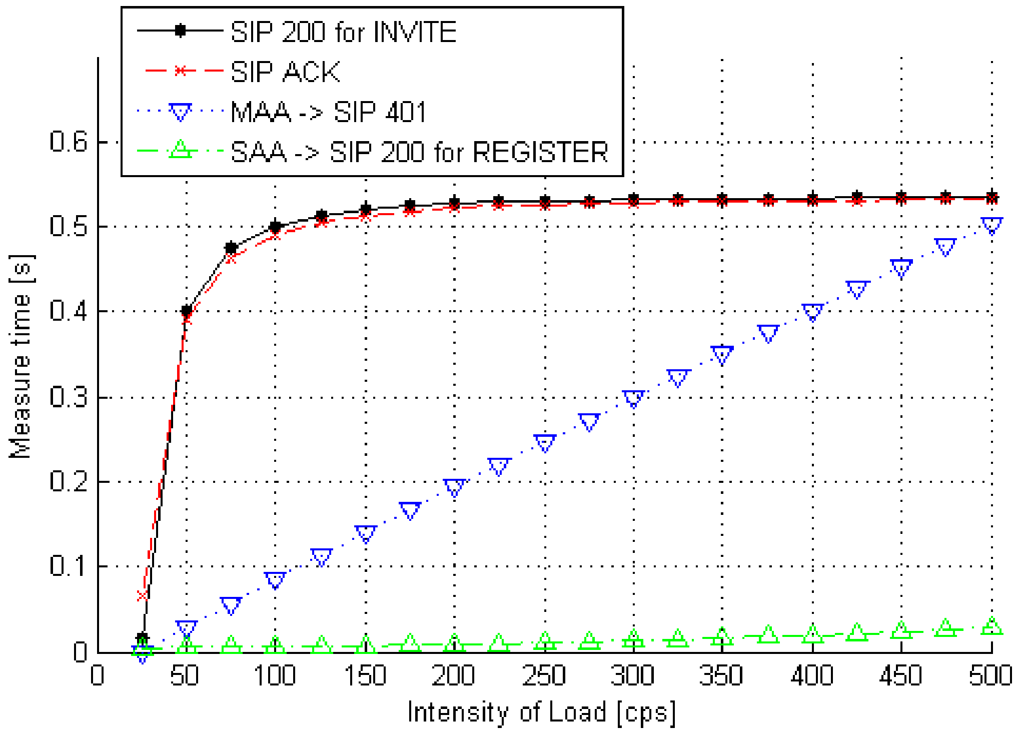


Figure 16. Response Time of SIP 200 and ACK at S-CSCF Drawn from Derived Functions.

this is not relevant in the production IMS network. From the perspective of queueing network, we can consider an IMS node as a finite buffer queue and a processing server. If the arrival rate (or traffic intensity) is higher than the server's processing rate, the queue buffer is saturated. This affects the response time of a message. Thus we can say that under a real-time condition with heavy usage the response time depends on all network components.

In order to overcome these issues, we propose a new analysis model where we also consider each IMS network as an M/G/1 queueing network. Compared to the previous methods where the response time for each SIP message type is measured at every IMS node, we measure the response time for messages (in general) for each node. This will take less effort and less time. Our model would solve the issue of scalability and provides general functions that can adapt to the production network. When more nodes are added, which is the case of a production network, our model can be expanded to accommodate this. Because we consider all IMS nodes, our calculations are more precise. The proposed model is introduced in Section 3.2. We then evaluate the response time at P-CSCF, I-CSCF, S-CSCF and HSS in our test-bed using this model in Section 4.6. Furthermore, our model can also evaluate the IMS saturation point and CPU utilization for each IMS component in Section 4.7.

3.2 Proposed Analysis

In this section we introduce a method to classify SIP and Diameter messages into different classes. The requests of a specific class are processed differently from other classes in each IMS node. Then we will use queueing theory with the well-known Pollaczek-Khinchine formula to evaluate the response time for each IMS node.

3.2.1 Registration Procedure Analysis Model

The IMS registration procedure was introduced in Section 2.3. This procedure contains many stages of receiving and forwarding messages between CSCFs and HSS. Instead of considering each message separately, we group them into classes. Each class has its own characteristics as shown in Table 5. They will help the the measurements of request routing independent from the node. Then we make some assumptions for our analysis model. We assume that the traffic flow follows a Poisson probability distribution which describes the average rate occurring in a fixed interval and timely independent of the previous event. For example, the average rate is 30 calls per second on average. However, it might fluctuate: sometimes greater than 30, sometimes smaller than 30 and once in a while to be 0. Given an average rate and a specific period of time, the Poisson distribution specifies how likely the rate would be. We also assume that the message processing time does not depend on the message type, which means that all messages are treated equally, and no priority is given for any certain message class. In our model, we neglect the probability of message loss. Message retransmission will not be considered.

The operation of message classes is shown in Figure 17 for the first registration attempt from the subscriber. Each IMS component is represented by a buffer and a server. Messages of a certain class enter the queue and wait for the server to process it. After being processed, the message can either remain in its class or get transformed to another class.

In the implementation of IMS system, HSS uses MySQL to manage the subscriber

Class Nomination	***Description***
1	Class 1 corresponds to the processing and transmitting REGISTER Request.
2	Class 2 corresponds to the processing and transmitting User Authentication Request (UAR).
3	Class 3 corresponds to the processing and transmitting HSS query to MySQL Database.
4	Class 4 corresponds to the processing and transmitting MySQL Responses to HSS.
5	Class 5 corresponds to the processing and transmitting User Authentication Answer (UAA).
6	Class 6 corresponds to the processing and transmitting Multimedia Authentication Request (MAR).
7	Class 7 corresponds to the processing and transmitting Multimedia Authentication Answer (MAA).
8	Class 3 corresponds to the processing and transmitting 401 Unauthorized message.
9	Class 3 corresponds to the processing and transmitting 200 OK message.

Table 5. IMS Request Classifications.

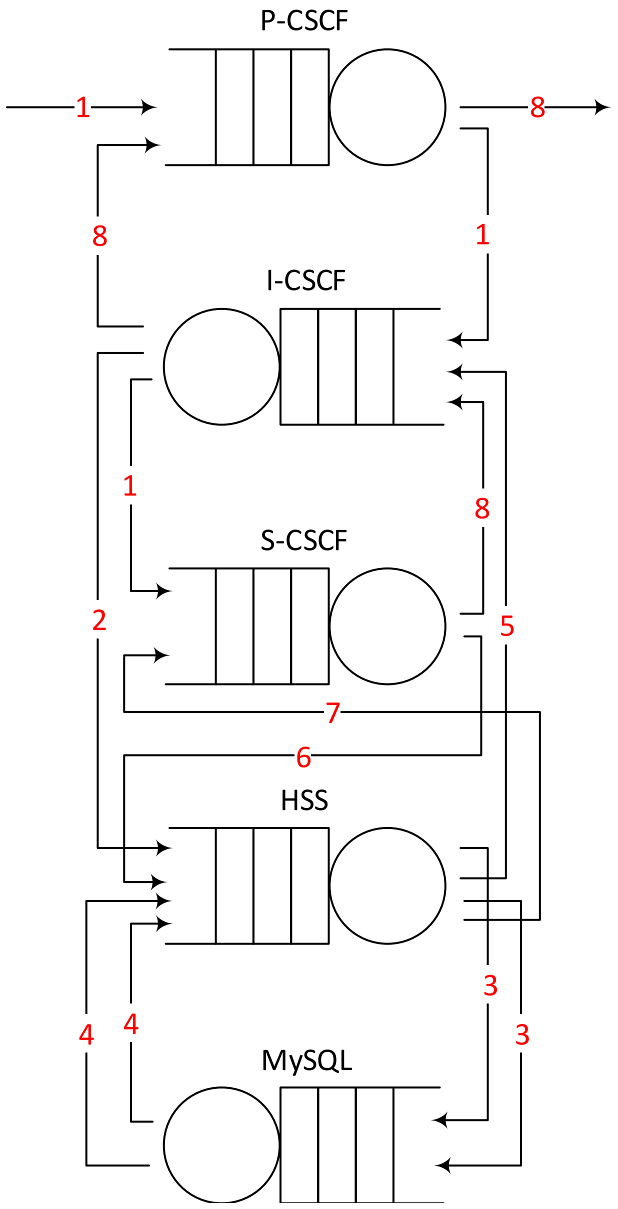


Figure 17. Message Classes Functionality of First Attempt Registration.

database. So we add MySQL as a component to our IMS performance analysis. An illustration describing the procedure in Figure 17 is shown below:

- A request of class 1, which corresponds to the REGISTER SIP message sent from a User Equipment, enters the network at a P-CSCF node.
- This class 1 request arrives at P-CSCF queue and is processed by the server. It still remains in class 1 after processing. P-CSCF then forwards it to I-CSCF.

- I-CSCF processes the received request and changes it to class 2 before sending to HSS.
- At HSS, class 2 query is transformed into class 3 which is MySQL database query. It is then forwarded to MySQL.
- After querying the database, the query is sent out with class 4 back to HSS.
- HSS transforms the class 4 request to class 5, and then sends to I-CSCF.
- Receiving class 5 request, ICSCF now knows which S-CSCF is used to serve the subscriber. Thus it transforms this class 5 to class 1 again and sends to S-CSCF for the registration.
- S-CSCF asks the HSS for the Authentication Information Request (MAR) by sending out class 6 request.
- The HSS again performs database query using class 3 and receives response of class 4.
- Once obtaining the results, the HSS transforms class 4 to class 7 to reply to the S-CSCF with Authentication Information Answer (MAA).
- The S-CSCF finally transforms the request to class 8 which is 401 Unauthorized, and sends it to I-CSCF.
- This class 8 request goes all the way through P-CSCF to the User Equipment.

In the general registration procedure described in Section 2.3, there are actually three phases: Unauthenticated Registration Attempt, IPsec Security Association Establishment and Authenticated Registration Attempt. However, in the test-bed that we built for this model, we only consider two registration attempts because the IPsec Association procedure accounts for a really short amount of time compared to the whole system delay time and it happens only once when the UE first registers. The first attempt is Unauthenticated Registration in which the IMS challenges the UE for the right authentication info, and the second one is Authenticated Registration in which the UE responds to the challenge. The class flow of the first attempt is described in detail as above. The second attempt is similar, with the class 8 request being replaced with class 9 which is 200 OK.

3.2.2 Mathematical Analysis

Before proceeding to the analysis, we have to decide what type of queueing network model we would use. The decision is important because the selected model must have the characteristics of the IMS system. According to our assumption, the incoming traffic is exponential distribution. However, the service time is not exponentially distributed. Therefore we would analyze the behaviours of the nodes using an M/G/1 queuing system.

We now use the concept of Open Queue Network with Different Classes of Customer as mentioned in [22] to build our model. The model contains an arbitrary but finite number of nodes and customer classes. We denote the set of IMS nodes as $M = \{1, ..., m\}$. In the production IMS system where the size of network grows enormously, the value of m can be more than 100. In our test-bed, only four nodes are built for the evaluation. Including the MySQL we have $m = 5$. The value of m does not necessarily correspond to the number of physical nodes. It is more linked to the functional blocks of the system. We then denote the set of system procedures as $P = \{1, ..., p\}$. A system procedure is defined as a process starting from the time a request enters the system until the time an expected response is received by the sender. For example, the live IMS network has many concurrent procedures such as registration, re-registration, de-registration, messaging, audio/video call, data retrieval, etc. Each of them will be counted in the set of procedures P. Last but not least, we denote the set of request class as $R^k = \{1, ..., R_k\}$. We have 9 classes as shown in Table 5, which means $k = 9$.

In a procedure $p \in P$, a request of class $r \in R^k$ at node $i \in M$ will be named $\{i, r\}^p$. Or we can say that the request is of state $\{i, r\}^p$. The set of all types of request for a certain procedure is written as:

$$S^p = \left\{(i, r)^p \mid i = \overline{1, m} \quad and \quad r = \overline{1, R_k}\right\} \tag{1}$$

In our open queueing model, the request arrives at the network according to the Poisson distribution. Let λ_0^p be the external arrival rate of procedure $p \in P$. The external request might arrive in a state $\{i, r\}^p$ with the probability of q_{ir}^p such that $\sum_{i \in M} \sum_{r \in R^k} q_{ir}^p = 1$. We form a row-vector consisting all of these probabilities which is defined as:

$$\boldsymbol{Q}^p = (q_{ir}^p) \mid i = \overline{1, m} \quad and \quad r = \overline{1, R_k} \tag{2}$$

From the above external arrival rate and probabilities (2) we can find the external flux as:

$$\lambda_0^p \times \boldsymbol{Q}^p \tag{3}$$

In the following part we will find the internal flux for the network. A request travels through the IMS network and changes its class with a transition probability. Let us say the request $\{j,s\}^p$ after being processed is sent to node i and is transformed to class r. The request becomes $\{i,r\}^p$ with the transition probability of $\theta^p_{js,ir}$. Then if we put all these transition probabilities together we can obtain the routing matrix for the procedure $p \in P$ as:

$$\boldsymbol{\Theta}^p = (\theta^p_{js,ir}) \mid i,j = \overline{1,m} \quad and \quad r,s = \overline{1,R_k} \tag{4}$$

We denote λ^p_{ir} as the internal arrival rate to state $\{i,r\}^p$. We can form another row-vector:

$$\boldsymbol{\lambda}^p = (\lambda^p_{ir}) \mid i = \overline{1,m} \quad and \quad r = \overline{1,R_k} \tag{5}$$

In the long-run when the system is stable, the queue length does not grow infinite. Every request entering a state will leave it eventually. Thus the long-run departure rate from state $\{i,r\}^p$ must be the same as its long-run arrival rate. We then can find the internal flux from the internal transition probabilities (4) and internal arrival rate (5) as:

$$\boldsymbol{\lambda}^p \times \boldsymbol{\Theta}^p \tag{6}$$

From the Equation (3) and (6) we can find the traffic rate at a node by summing the external and internal flux:

$$\boldsymbol{\lambda}^p = \lambda^p_0 \boldsymbol{Q}^p + \boldsymbol{\lambda}^p \boldsymbol{\Theta}^p \tag{7}$$

Given the rates for all classes and procedures as shown in the matrix of (7) we can calculate the arrival rate of requests for a certain node i by summing all of them:

$$\lambda_i = \sum_{p=1}^{p} \sum_{r=1}^{R_k} \lambda^p_{ir} \tag{8}$$

Let us consider IMS as an open domain with multiple subsystems (each IMS component is considered a subsystem). Figure 17 depicts a situation where a stream of equivalent requests comes into a subsystem, remains inside the subsystem and then leaves after a finite time. If we consider one request, it moves in and out of one subsystem many times. Each results in a transit time which is the time between the entrance and the exit. The sum of these transit time is sojourn time of the subsystem for that particular request. The significance of sojourn time can be seen in the following explanation. If the stream of request coming to the IMS system is constant, it will reach the steady state. It can be shown that the number of requests on the subsystem is equal to the stream of requests into the system time the mean sojourn time of the subsystem. According to Little's theory [13] , it is called the occupancy principle:

$$requests\ in\ subsystem = (requests\ into\ system) \times (sojourn\ time\ of\ subsystem)$$

Since the requests are processed in their arriving order, the sojourn time is the sum of two stages: waiting time and service time. If the system is empty when the request comes, the waiting time is zero. Thus the sojourn time is equal to the service time. The sojourn time at node i can be calculated by the Pollaczek-Khinchine formula for M/G/1 queueing system as [13]:

$$\nu_i = \overline{x}_i + \frac{\lambda_i \overline{x}_i^{(2)}}{2(1-\rho_i)} \tag{9}$$

where ν_i is the sojourn time at the node, $\overline{x}_i$ is the average serving time for each request at the node i, $\overline{x}_i^{(2)}$ is the second moment of distribution function of the request serving time at node i and ρ_i is the traffic intensity which is defined as:

$$\rho_i = \frac{\lambda_i}{\mu_i} = \lambda_i \overline{x}_i \tag{10}$$

The second moment of the request service time $\overline{x}_i^{(2)}$ is [13]:

$$\overline{x}_i^{(2)} = \frac{1}{\mu^2} = \overline{x}^2 \tag{11}$$

By substituting Equation (10) and (11) into Equation (9) we finally have the sojourn time expressed in terms of serving time ($\overline{x}_i$) and arrival rate (λ_i):

$$\nu_i = \overline{x}_i + \frac{\lambda_i \overline{x}_i^2}{2(1-\lambda_i \overline{x}_i)} \tag{12}$$

Now considering the IMS nodes function independently from each other, we can calculate the average processing time for each procedure as the sum of the sojourn time of the requests at the nodes. The general formula is:

$$\nu = n\nu_{P-CSCF} + n'\nu_{I-CSCF} + n''\nu_{S-CSCF} + n^{(3)}\nu_{HSS} + n^{(4)}\nu_{MySQL} \tag{13}$$

where n, n', n'', $n^{(3)}$ and $n^{(4)}$ is the number of messages processed by each IMS node.

Referring to Figure 17 we can deduce the value for the first registration attempt. The second registration attempt follows the same process and has the same number of requests. Thus we have to add up messages at each IMS node as shown in Table 6.

We are evaluating only the registration procedure, which means that $p = 1$. Thus the response time for registration can now we rewritten from Equation (13) as:

$$\nu_{reg} = 4\nu_{P-CSCF} + 6\nu_{I-CSCF} + 4\nu_{S-CSCF} + 8\nu_{HSS} + 4\nu_{MySQL} \tag{14}$$

	P-CSCF	***I-CSCF***	***S-CSCF***	***HSS***	***MySQL***
1st attempt	2	3	2	4	2
2nd attempt	2	3	2	4	2
Total	4	6	4	8	4

Table 6. Number of Requests in Each IMS Node.

3.2.3 Call Setup and Termination Analysis

Figure 43 in Appendix B shows the call setup procedure for two subscribers. This is summarized in Table 7. At each step, there are two messages passing through the P-CSCF, one message for the I-CSCF and two for the S-CSCF. We have a total of six steps, thus the response time for the call setup is:

$$\nu_{setup} = 12\nu_{P-CSCF} + 6\nu_{I-CSCF} + 12\nu_{S-CSCF} \tag{15}$$

Figure 44 in Appendix B shows the IMS call termination procedure. There are two steps in this procedure. The first step is to send the BYE message from UE1 to UE2 and the second step is to send the 200 OK from UE2 to UE1. When the UE1 receives the 200 OK message the call is terminated. The total number of messages passing through the P-CSCF is four and through the S-CSCF is two. Thus, the response time for the call termination is:

$$\nu_{terminate} = 4\nu_{P-CSCF} + 2\nu_{S-CSCF} \tag{16}$$

For a simple IMS system with only the registration, call setup and call termination procedure we can have the system response time as:

$$\nu = \nu_{reg} + \nu_{setup} + \nu_{terminate}$$

Step	***Message Type***	***Direction***
1	INVITE	UE1 TO UE2
2	180 Ringing	UE2 TO UE1
3	PRACK	UE1 TO UE2
4	200 OK	UE2 TO UE1 for User Ringing
5	200 OK	UE2 to UE1 for User Answer
6	ACK	UE1 to UE2

Table 7. Call Setup Procedure Steps.

In this chapter, we have analyzed the previous work and presented their drawbacks. We then proposed our model which uses a queueing network to evaluate response time for registration, call setup and termination procedure. The obtained functions will be used in Chapter 4 with measured service time to calculate the response time for each procedure. Furthermore, this model can also be extended to include other services such as media retrieval, presentation, etc. Each procedure can be evaluated separately to see how the IMS system reacts for a certain service. The results can also be added together to obtain the total performance.

Chapter 4

Testing and Evaluating

In this chapter, we implement the IMS system for our test-bed. Then we will use the IMS Bench SIPp application to generate traffic and feed into the IMS system. The performance results will be compared to the numerical analysis in our proposed model.

4.1 Introduction to Test-bed Environment

At the time this document is written, Open Source IMS Core System is the only open source testing environment available. Quite many R&D projects have been using this system for functional and performance evaluations. This IMS Core System is developed by Fraunhofer Institute FOKUS with the purpose of implementing IMS with a flexible and extensible solution. It will help building initial concepts about IMS components. Based upon the conditions mentioned above, future research concepts can be developed.

The Open Source IMS core (Figure 18) contains three CSCFs (including P-CSCF, I-CSCF and S-CSCF) and one HSS. These CSCFs are built upon the SIP Express Router (SER) which is an open-source SIP server supporting many features of RFC 3261. These features include various subscription database backends (MySQL, Oracle, PostgreSQL), management features (remote management via XML-RPC, load-balancing), NAT traversal, telephony features (LCR, speeddial), multidomain hosting, ENUM, presence, etc. [23]. The Home Subscriber Server (HSS) is built upon MySQL to manage subscription profiles and policies. These open-source components can be implemented on either separate physical/virtual machines or together on one physical/virtual machine. Normally in an R&D testing environment, each component is located on different powerful servers having dedicated IP addresses. Because of resource limitation we have installed them on one PC using the loop-back address 127.0.0.1 and different port numbers for each component. The IMS

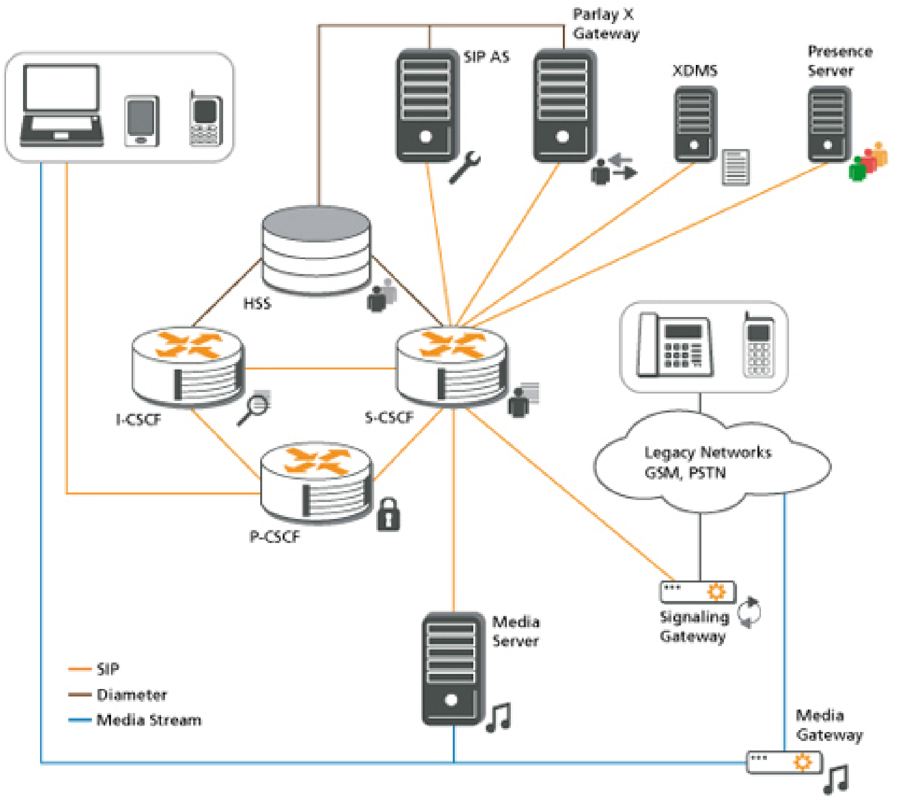


Figure 18. Open Source IMS Core Playground.[2]

core can be installed on different Linux distributions such as Redhat, Debian, CentOS, etc. We choose Ubuntu distribution as it is more user-oriented and suitable for our small test-bed. Our test-bed uses Ubuntu version 14.04 Long Term Support (LTS) because it is the most stable at the current time.

In order to test the basic functionality of IMS, we use the SIP client called myMONSTER (Multimedia Open InterNet Services and Telecommunication EnviRonment)[24]. This client is also developed by Fraunhofer FOKUS. One advantage when using this client is that it is compatible very well with the Open Source IMS Core as it is from the same creator. myMONSTER has two versions: desktop and mobile. This means that the Service Provider can test their IMS solutions with both their wire and wireless networks. Most other clients lack this feature. Furthermore, the author also provides development tool-kits which will help to create extensive widgets and add-ons to test IMS features.

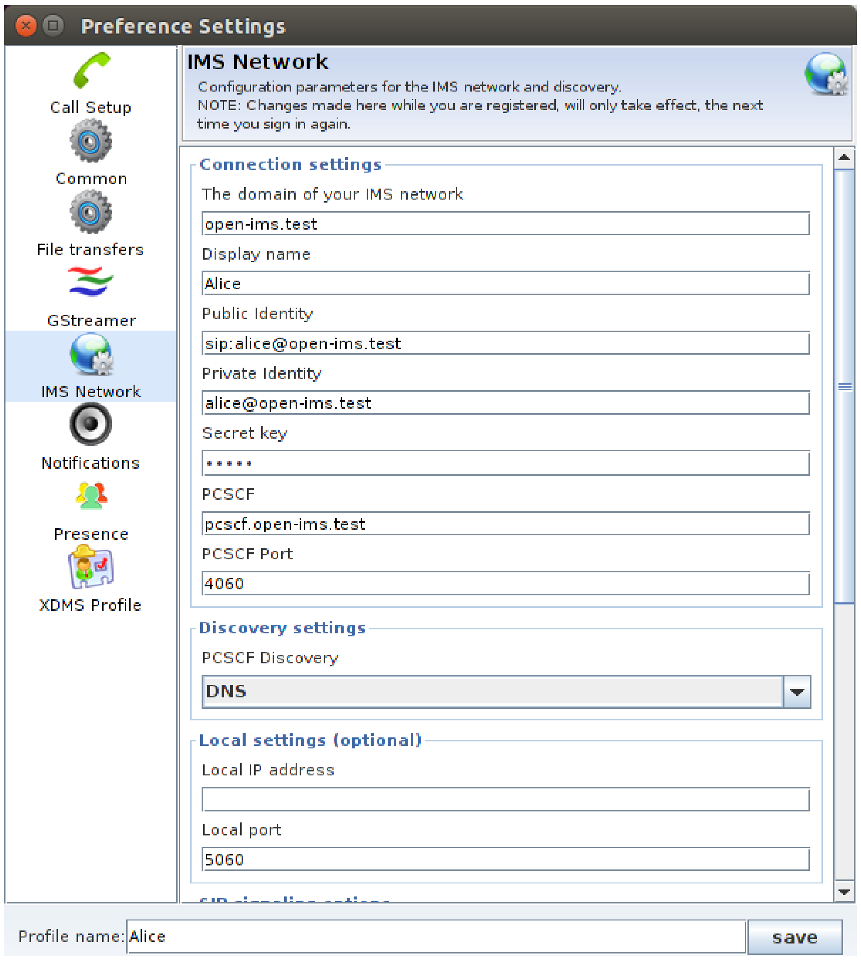
To evaluate system performance we have to create multiple requests and feed them to the IMS test-bed. This can be achieved by using IMS Bench SIPp. This is a traffic generator providing an open source implementation of the IMS/NGN Performance Benchmark, ETSI specification TS 186 008 [3]. The application includes one SIPp manager and many SIPp clients. The SIPp manager will control the SIPp clients to run the multiple test cases specified in XML-format files. The performance results are saved at SIPp clients. The SIPp manager collects the results only after the running phases. The reports can be generated from these results.

4.2 OpenIMS Implementation

Before implementing the Open Source IMS Core system, we would like to explain the two installation flavours that we can use for our test-bed system. The first one is the installation of the components on physical and fixed servers. This is the method most researchers use to build their systems. We also use this approach in our test-bed. The second one is the installation on Virtual Machines (VM). We believe that VM will become a major trend in the near future. Even though physical installation provides efficiency in processing speed, the provisioning takes more time than it does on a VM. In the VM environment, the image clone capability provides an easier installation and faster provision. We have tried to install P-CSCF with VMWare Workstation in our test-bed. We then managed to clone P-CSCF to I-CSCF, S-CSCF and HSS. These four components run as four separate servers under VMWare Controller. But due to the resource limitations of our workstation, high performance could not be made in this system. In the R&D testing environment, many telecommunication equipment manufacturers such as Alcatel-Lucent, Juniper and especially Cisco can provide the best solutions combining VMware and Data Center Infrastructure suitable for IMS provisioning. However, the performance of IMS in virtual environment still remains as an open question for future research. The IMS installation steps for physical machine are shown in Appendix C.

4.3 IMS Functional Test

Before starting the IMS performance evaluation, we have to make sure that our IMS test-bed is working as expected. We use the SIP client myMONSTER as mentioned in the introduction for basic messaging, audio and video call. The installation is shown in Appendix F. The FHoSS database comes with two sample users named Alice and Bob. We will use these users for our testing purpose. A profile configuration for Alice is shown in Figure 19. The messaging and call tests are shown in Figure 20 and Figure 21 .



IMS Network

Configuration parameters for the IMS network and discovery.
NOTE: Changes made here while you are registered, will only take effect, the next time you sign in again.

Connection settings

The domain of your IMS network: open-ims.test

Display name: Alice

Public Identity: sip:alice@open-ims.test

Private Identity: alice@open-ims.test

Secret key: •••••

PCSCF: pcscf.open-ims.test

PCSCF Port: 4060

Discovery settings

PCSCF Discovery: DNS

Local settings (optional)

Local IP address:

Local port: 5060

Profile name: Alice

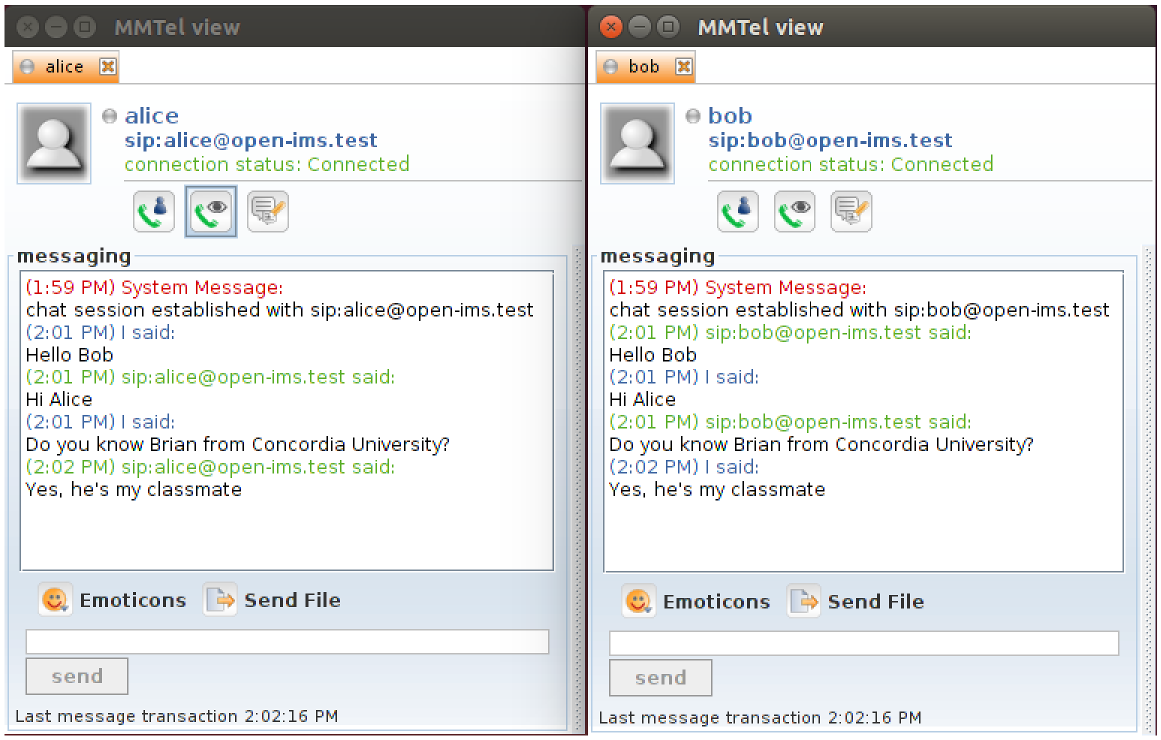
Figure 19. Alice Profile Configuration.

4.4 IMS Bench SIPp Implementation

In this section, we explain the general structure and installation steps of IMS Bench SIPp tool. Then we will perform the registration test using the XML-scenario test cases.

4.4.1 Component Structure

Originally, SIPp is a SIP client for simple User Agent Client (UAC) and User Agent Server (UAS) testing. However, as the need for an IMS performance evaluation increases, more complicated test scenarios are needed. SIPp has evolved into a separate branch for IMS testing. It is IMS Bench SIPp. The tool includes one SIPp Manager and one or more SIPp Instances (Figure 22). Each SIPp component (Manager or Instances) can run on different physical machines or on one machine. The SIPp Manager contains all scenarios and the ratio of each scenario to test. It communicates with SIPp Instance over TCP. The SIPp


MMTel view

alice

sip:alice@open-ims.test

connection status: Connected

messaging

(1:59 PM) System Message:
chat session established with sip:alice@open-ims.test
(2:01 PM) I said:
Hello Bob
(2:01 PM) sip:alice@open-ims.test said:
Hi Alice
(2:01 PM) I said:
Do you know Brian from Concordia University?
(2:02 PM) sip:alice@open-ims.test said:
Yes, he's my classmate

Last message transaction 2:02:16 PM

MMTel view

bob

sip:bob@open-ims.test

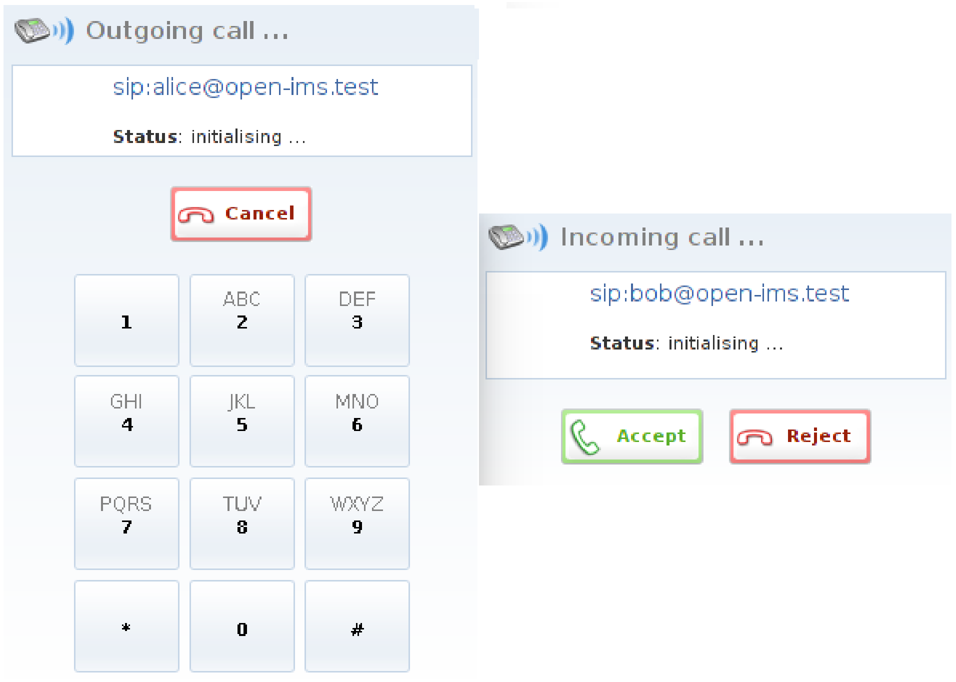
connection status: Connected

messaging

(1:59 PM) System Message:
chat session established with sip:bob@open-ims.test
(2:01 PM) sip:bob@open-ims.test said:
Hello Bob
(2:01 PM) I said:
Hi Alice
(2:01 PM) sip:bob@open-ims.test said:
Do you know Brian from Concordia University?
(2:02 PM) I said:
Yes, he's my classmate

Last message transaction 2:02:16 PM

Figure 20. myMonster Messaging Test.


Outgoing call ...

sip:alice@open-ims.test

Status: initialising ...

Incoming call ...

sip:bob@open-ims.test

Status: initialising ...

Figure 21. myMONSTER Call Test.

Instance receives the full set of scenarios from the SIPp Manager and executes them with its own set of users. The SIPp Instances communicate with each other over TCP to exchange user reservations and timing data.

The working principle for IMS Bench SIPp is explained below:

- SIPp Manager is launched first. It is loaded with testing scenarios along with the ratio of each scenario. This is a nice feature of this tool. We can specify this ratio according to the ratio in the production network for a better evaluation. SIPp manager is now waiting for connections from SIPp Instances.
- SIPp Instances are launched and connect to the SIPp Manager. SIPp Manager then sends the full set of scenarios including scenario ratio to SIPp Instances.
- SIPp Instances execute the scenarios with its own set of users. In order not to affect the performance results, all the measurements are stored locally where SIPp Instances run. Only selected management and summary information is sent back to SIPp Manager.
- SIPp Manager displays the real-time summary received from SIPp Instances.
- After the run, a post-processing tool (at SIPp Manager) fetches all generated data from SIPp Instances and generates a report.
- A monitoring agent called CpuMem can also be run on the SUT to collect system statistics. It communicates and provides the statistics to the SIPp Manager over TCP.

4.4.2 Installation

The installation process can be divided into two steps. First we need to install all the prerequisites required for the tool. Because we will run test cases randomly with many open network connections, we have to adjust the system limit for these requirements. Furthermore, we adjust the kernel Timer Frequency to 1000Hz in order to achieve approximately millisecond precision (the current frequency for Ubuntu is 250Hz.) When having all the prerequisites we can proceed to install IMS Bench SIPp. The installation steps are shown in Appendix D.

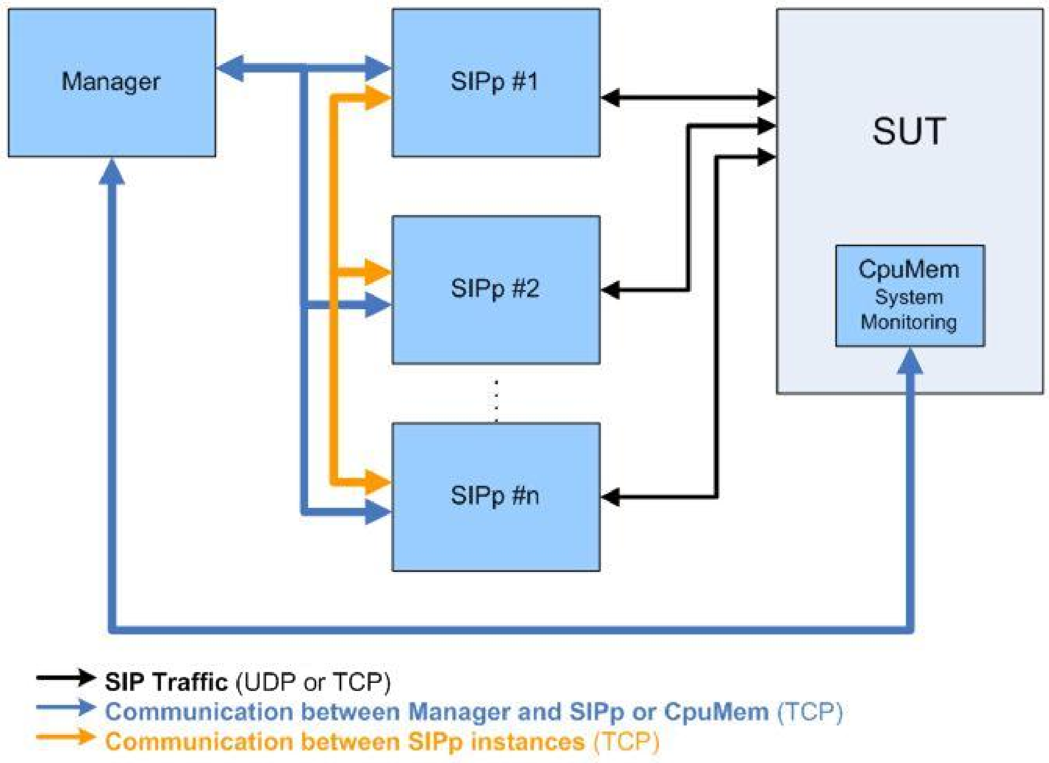

Manager
SIPp #1
SIPp #2
SIPp #n
SUT
CpuMem
System
Monitoring
SIP Traffic (UDP or TCP)
Communication between Manager and SIPp or CpuMem (TCP)
Communication between SIPp instances (TCP)

Figure 22. IMS Bench SIPp Components.[3]

4.4.3 Test Cases

General Concepts

The most important feature of IMS Bench SIPp is multiple scenario support. The scenarios are stored at the SIPp Manager. When the SIPp Instances connect to the Manager, they download the scenarios and run them according to the XML set-up configuration. There are two types of scenarios, client-side and server-side. The client-side scenario starts when the SIPp Instance executes its command. In one test case, their might be more than one client-side scenario. Each of them has its own distribution and ratio. The Instance will select them randomly. The server-side scenarios are activated only when receiving a preparation signal from the SIPp Instance. So we can say that client-side scenario is active and server-side is passive. An example of IMS Bench Setup is shown in Figure 23. The SIPp Instances communicate with the SIPp Manager (192.168.2.2) through port 5000. They communicate and monitor SUT (192.168.2.4) through port 4060. Each SIPp Instance has its own user set (*ims_user_1.inf* and *ims_user_2.inf*) to execute the scenarios. The SIPp Instances communicate with each other using non-SIP message to negotiate user reservations and timing parameters. We assume that user Alice wants to call user Bob in this example. SIPp Instance 1 initiates a client-side scenario and generates the traffic to System Under Test (SUT). We expect that the INVITE message will go through SUT and get to SIPp Instance 2. Instance 1 sends a preparation signal to Instance 2 and asks it to start server-side scenario which will receive and respond to the INVITE message. The following steps will explain how to start the test case for Figure 23.

- **SIPp Manager**

```
user@ubuntu# ./manager -f manager.xml
```

  The Manager starts with the configuration described in file manager.xml. Details are explained in Appendix I.

- **SIPp Instance 1**

```
user@ubuntu# ./sipp 127.0.0.1:4060
  -id 1
  -i 192.168.2.21
  -user_inf ./ims_users_1.inf
  -rmctrl 192.168.2.2:5000
  -trace_err -trace_cpumem -trace_scen -trace_retrans
```

- **SIPp Instance 2**

```
user@ubuntu# ./sipp 127.0.0.1:4060
  -id 2 -i 192.168.2.22
  -user_inf ./ims_users_2.inf
  -rmctrl 192.168.2.2:5000
  -trace_err -trace_cpumem -trace_scen -trace_retrans
```

  where

-id	The ID of the SIPp Instance.
-i	The IP of the SIPp Instance.
-user_inf	The user set that the SIPp instance will use for testing. Each instance has its own set. This set can be generated using the script /opt/ims_bench/user_gen.pl which is explained in Appendix H .
-rmctrl	The IP and port of the SIPp Manger. Port is 5000.
-trace_err	Save all unexpected messages in <scenario file name>_<pid>_errors.log.

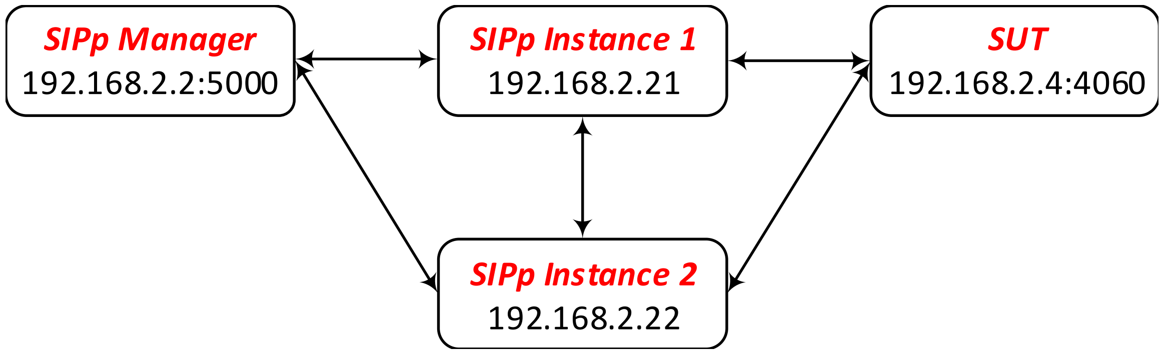


Figure 23. IMS Bench SIPp Example.

-trace_cpumen	Save the CPU/MEM per second in sipp_<pid>_cpumem.csv.
-trace_scen	Save scenario execution, result and response time in sipp_<pid>_scen.csv.
-trace_retrans	Save number of retransmission per second in <scenario_name>_<pid>_retrans.csv.

- **SUT**

```
user@ubuntu# ./cpum 192.168.2.2:5000
```

Test Case for our IMS test-bed

Before starting IMS Bench we need to have enough users in FHoSS. We created a script named **multiusers.sh** (see Appendix J) to add multiple users and their subscription information to the database. For the purpose of testing registration procedure, we add 6000 users to FHoSS. We then can use the script **user_gen.pl** to generate a user list **ims_users_1.inf** for IMS Bench. In our test-bed, all the IMS components and IMS Bench SIPp are running on the one machine using the loopback interface **127.0.0.1**. Thus to start the performance testing we use different parameters:

- **SIPp Manager**

```
user@ubuntu# ./manager -f manager_reg.xml
```

- **SIPp Instance**

```
user@ubuntu# ./sipp 127.0.0.1:4060
  -id 1
  -i 127.0.0.1
  -user_inf ./ims_users_1.inf
```

```
-rmctrl 127.0.0.1:5000
-trace_err -trace_scen -trace_retrans
```

After launching the SIPp instance, go back to the SIPp Manager, we can see that the instance is connected. Press "**e**" to start the testing.

In the above command, besides different IP parameters, we also use our own manager configuration file (**manager_reg.xml**) and test scenario (**test_reg.xml**). These are specifically for the registration procedure evaluation. Due to resource limitations, our test-bed can only handle maximum 30 calls per second (CPS). If the arrival rate is over 30 CPS, the Inadequately Handled Scenarios (IHS) ratio is really high and the measurement results are not precise. Appendix K describes the configuration and scenario XML files for our test-bed. The general registration procedure using the SIPp Instance is shown in Figure 24. The SIPp Instance will run with the users defined in the list **imsuser_1.inf**. Initially the users are assigned to the unregistered pool (number 0). When the instance picks a user for the registration, it moves the user to the pending pool (number 1). The instance then generates a REGISTER message and feeds it to the IMS test-bed. It expects to receive the 401 Unauthorized message from the IMS system. If it fails to receives the 401 message, the registration is considered unsuccessful. The user is moved back to the unregistered pool. The IHS ratio also increases due to this unsuccessful event. The procedure continues until the SIPp Instance receives the 200 OK message. The registration is now successful and the user is moved to registered pool (number 3).

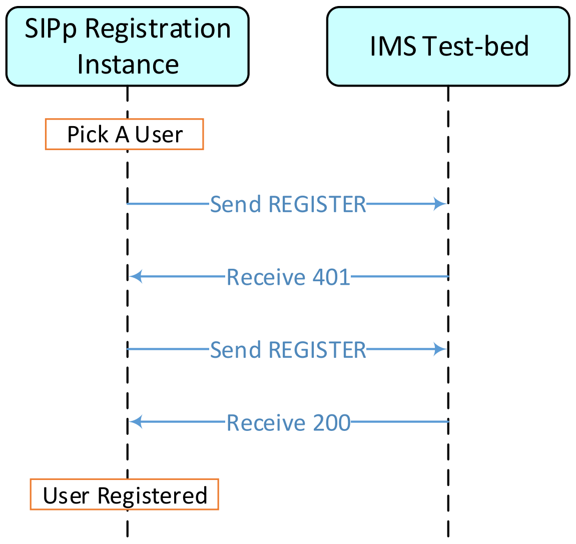


Figure 24. SIPp Registration Procedure.

4.5 Measured Performance Results

Following the instructions in Appendix L, we get the average response time for the first and second registration attempt. Then we can calculate the average time for IMS registration procedure as the sum of these two attempts. We measure the response time with different call rates for multiple times. An example for the response time of 30cps is shown in Figure 25. At the first few seconds of the measurement, the call rate increases from

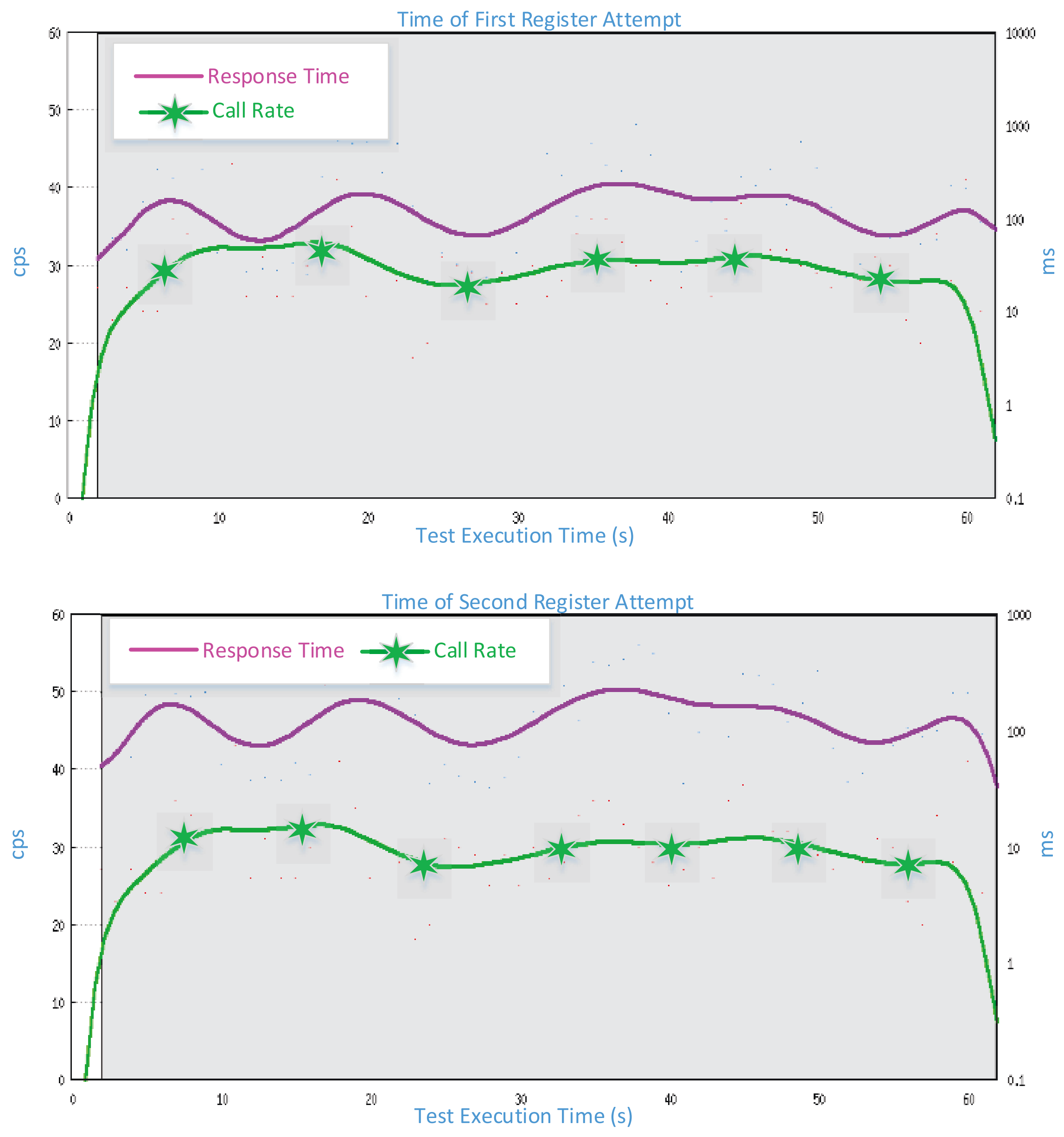


Figure 25. Response Time for IMS Test-bed with 30cps.

0 to approximately 30. This is the time for the requests to fill up the queue and is not considered in our evaluation. Then the response time fluctuates when the system warms up and the requests start circulating in the IMS system. After the warm up, the response time becomes more stable. At the end of the measurement, the call rate eventually drops to 0. The response time starts to fluctuate and to drop down. This time is not considered for the evaluation either. The average response time for the first and second attempt are $218.58ms$ and $193.14ms$. Thus the average response time for the whole process is $411.72ms$. As we mentioned in Appendix G, our test-bed has been adjusted with Ubuntu kernel frequency of 1000Hz to achieve around millisecond precision in scenario attempt scheduling and in timing measurements.

4.6 Response Time Evaluation

In this section, we will evaluate two test cases. In the first test case, only the registration scenario is specified. The measured and calculated response time will be compared. This test case will prove that our model can evaluate separate procedures. The second test case includes three scenarios: registration, call setup and call termination. The measured and calculated response time for each procedure are compared. This test case will prove that our model can adapt and include multiple procedures. In this second test case, we only consider the controlling traffic (SIP and Diameter). The media data (voice call) is transferred in the data plane and is not considered in our evaluation. Because of resource limitation, we did not implement media servers for video services. We only evaluate the registration and voice call services.

4.6.1 Test Case 1: Registration Procedure Only

Based on the classifications in Table 5 and the request flow in Figure 17, we defined the flow for all the states $\{i, r\}$ (because there is only registration procedure in our test case, $p = 1$) as in Figure 26. We number the CSCFs, FHoSS and MySQL with node number from 1 to 5. The *REGISTER* request (class 1) first comes to P-CSCF (node 1), which is state $\{1, 1\}$. Then it moves to I-CSCF (node 2), which is state $\{2, 1\}$. When it gets to FHoSS (node 4), the request changes to UAR (class 2) which is state $\{4, 2\}$. The procedure continues until state $\{3, 7\}$ where there are two possibilities. If it is the first registration attempt, the request changes to *401 Unauthorized* (class 8) and the state is $\{2, 8\}$. If it is the second registration attempt, the request changes to *200 OK* (class 9) and the state is $\{2, 9\}$. We now can form the transition probability matrix Θ for the 15 states in Figure 26. All the transition probabilities are 1 except the transition from state $\{3, 7\}$ to state $\{2, 8\}$

and $\{2, 9\}$ are 0.5.

From the Equation (7) we can get the arrival rate matrix as:

$$\begin{aligned} \boldsymbol{\lambda} &= \lambda_0 \boldsymbol{Q} + \boldsymbol{\lambda\Theta} \\ \boldsymbol{\lambda} - \boldsymbol{\lambda\Theta} &= \lambda_0 \boldsymbol{Q} \\ \boldsymbol{\lambda}(\boldsymbol{I} - \boldsymbol{\Theta}) &= \lambda_0 \boldsymbol{Q} \\ \Rightarrow \qquad \boldsymbol{\lambda} &= (\lambda_0 \boldsymbol{Q})(\boldsymbol{I} - \boldsymbol{\Theta})^{-1} \end{aligned} \tag{17}$$

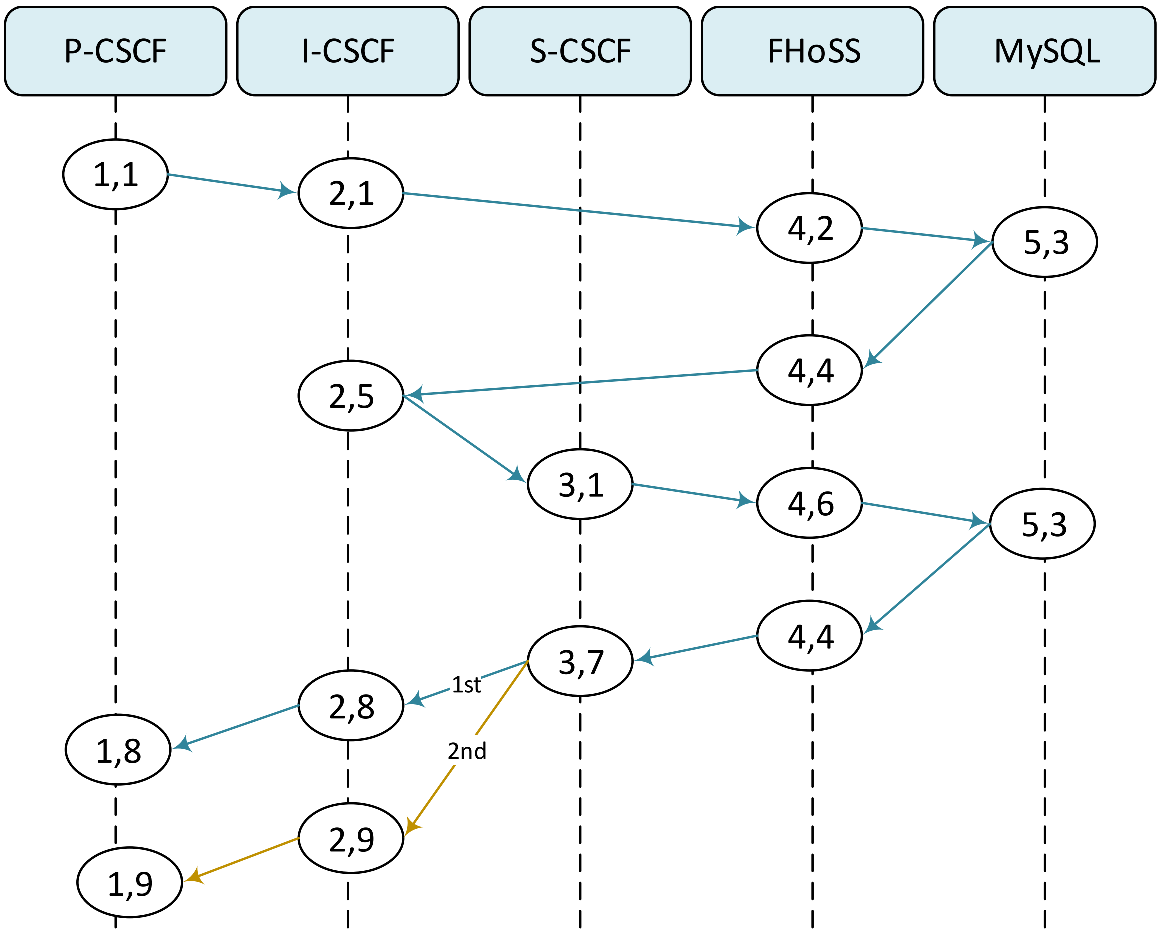


Figure 26. State Flow for Registration Procedure.

The transition probability matrix $\mathbf{\Theta}$ is

$$\mathbf{\Theta} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{18}$$

The external arrival rate is

$$\lambda_0 = 30\ cps \tag{19}$$

The external arrival probabilities are

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{20}$$

Substituting Equation (18), (19) and (20) into Equation (17), we get the transition rate matrix between the states as:

$$\boldsymbol{\lambda} = \begin{bmatrix} 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 & 15 & 15 & 15 & 15 \end{bmatrix}$$

We then can get the arrival rate for each node as (using the Matlab code **ims_eval.m** in Appendix M):

$$\lambda_{P-CSCF} = 60\ cps$$
$$\lambda_{I-CSCF} = 90\ cps$$
$$\lambda_{S-CSCF} = 60\ cps$$
$$\lambda_{FHoSS} = 120\ cps$$
$$\lambda_{MySQL} = 60\ cps$$

Besides the arrival rate (λ_i) we need the serving time ($\overline{x}_i$) to calculate the sojourn time at each node as in Equation (12). Because we run our IMS system on the loopback interface (127.0.0.1) we can use Wireshark to monitor the transmitted packets among the components. Figure 27 shows the associated ports with message flows between the IMS component in our test-bed. There are five ports that we need to pay attention to. For SIP message communication, they are 4060, 5060 and 6060 respectively for P-CSCF, I-CSCF and S-CSCF. For Diameter query, they are 3869 and 3870 respectively for I-CSCF and S-CSCF. In addition, the FHoSS queries the MySQL database using port 3306. The ports with XXX notation are random port numbers available in the IMS test-bed system. They are usually greater than 20000.

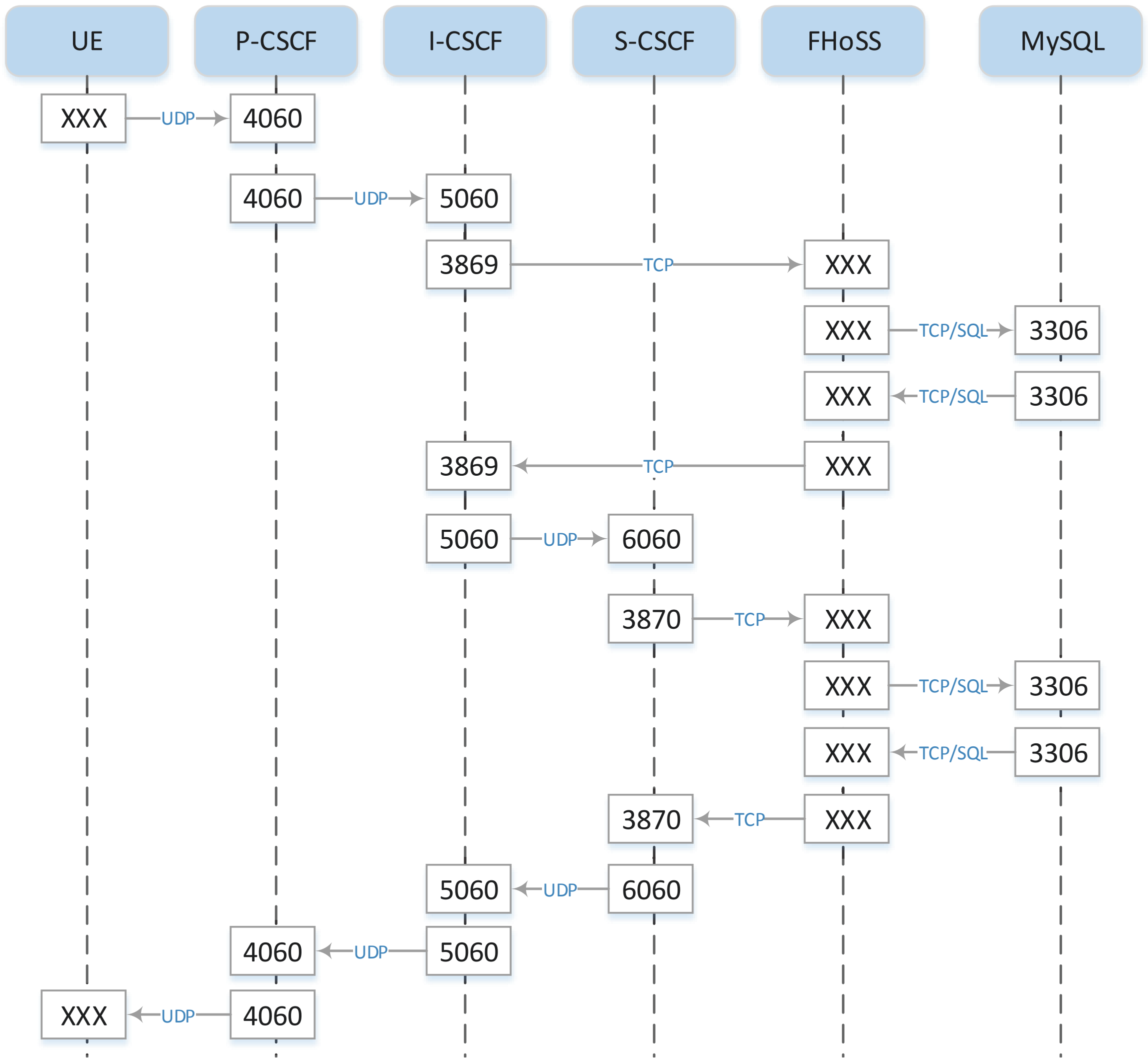


Figure 27. Request flow with associated ports.

The average request serving time in each node are recorded in Table 8. We perform the five measurements at four call rates: 30cps, 25cps, 20cps and 15cps. In each node, the request serving time is measured and in the last column the sojourn time is calculated using Equation (12). From these results, we can see that our test-bed system is not quite stable at 30cps rate, the sojourn time fluctuates in a wide range. However, when we decrease the call rate to lower values, the variance is little. This is illustrated in Figures 28, 29, 30, 31 and 32. Using the Matlab code in the Appendix P, we can calculate the confidence interval for each data set at each IMS node. At higher call rates, with the confidence of 98% we obtain a wider interval. For example, the results for P-CSCF are shown in Table 9. At 15cps, we are 98% confident that the measured results fall between 32.088ms and 32.956ms. At 30cps, we are 98% confident that the measured results fall in a wider range between 39.030ms and 47.933ms.

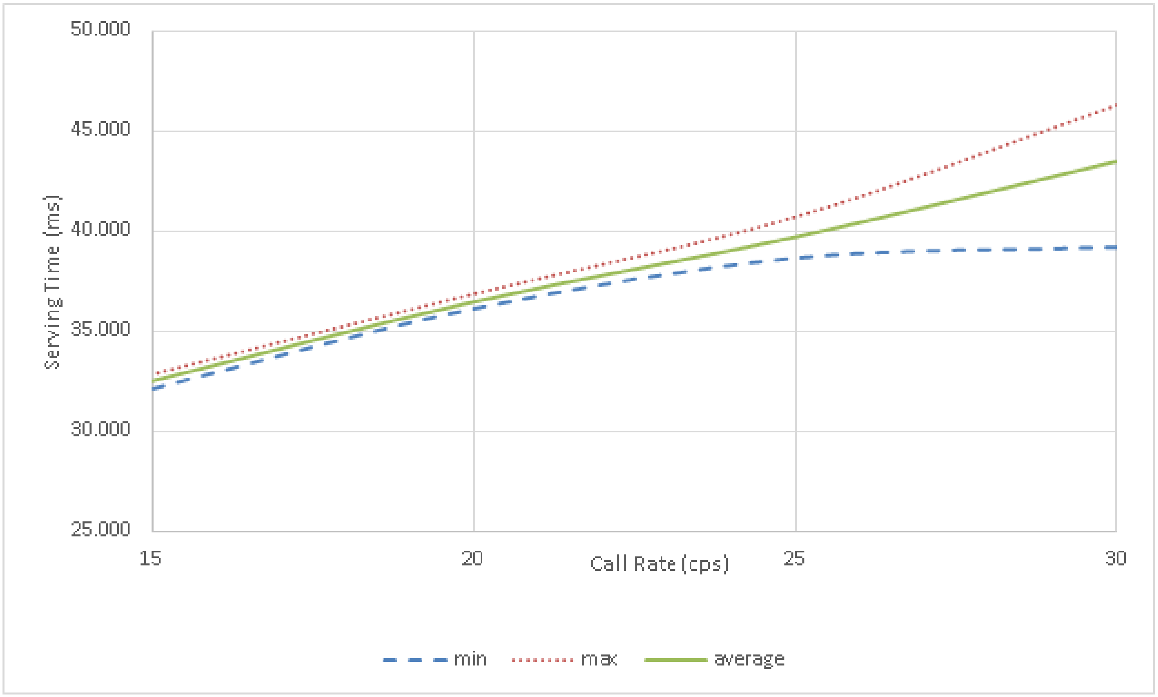


Figure 28. Serving Time at P-CSCF for Registration Procedure.

Besides measuring the request serving time at each IMS node, we also measure the average total response time for the registration procedure. The results are shown in the third column of Table 10. The second column is the calculated sojourn time extracted from Table 8. In the last column, we calculate the differences between the measured and calculated results. The differences are mostly over 1ms for 30cps rate, while they are less than 0.7 ms for lower call rate. In order to understand the differences between measured

Call Rate	$\overline{x}_{P-CSCF}$	$\overline{x}_{I-CSCF}$	$\overline{x}_{S-CSCF}$	$\overline{x}_{FHoSS}$	$\overline{x}_{MySQL}$	$\nu_{calculated}$
30cps	39.198	22.841	70.817	21.462	0.582	375.347
	44.691	28.102	69.079	22.243	0.602	401.804
	43.986	26.531	69.264	23.787	0.594	402.211
	46.326	29.763	73.498	28.688	0.611	438.693
	43.207	28.352	69.836	24.872	0.593	411.598
25cps	38.644	22.213	67.876	22.152	0.495	369.059
	39.124	25.452	63.268	22.813	0.516	373.206
	40.227	26.574	62.673	22.615	0.506	376.776
	39.782	25.113	60.552	22.791	0.488	367.927
	40.728	26.685	68.754	22.328	0.497	389.107
20cps	36.124	24.047	58.492	21.421	0.557	347.953
	36.312	24.127	58.145	21.073	0.623	346.623
	36.870	24.784	58.342	21.234	0.531	350.556
	36.295	24.341	58.470	21.514	0.485	349.361
	36.742	24.615	57.948	21.863	0.524	351.501
15cps	32.546	22.231	51.752	19.987	0.483	316.005
	32.129	21.827	51.421	20.262	0.492	314.397
	32.476	22.375	51.936	20.193	0.434	317.431
	32.842	22.634	51.048	20.117	0.539	317.006
	32.617	22.268	51.524	20.952	0.518	319.712

Table 8. Average Serving Time (ms) for Registration Procedure.

Call Rate	*Mean (ms)*	*CI Interval (ms)*
30cps	43.482	39.030 - 47.933
25cps	39.701	38.302 - 41.100
20cps	36.469	35.933 - 37.005
15cps	32.522	32.088 - 32.956

Table 9. Confidence Interval for P-CSCF Serving Time at 98% Confidence.

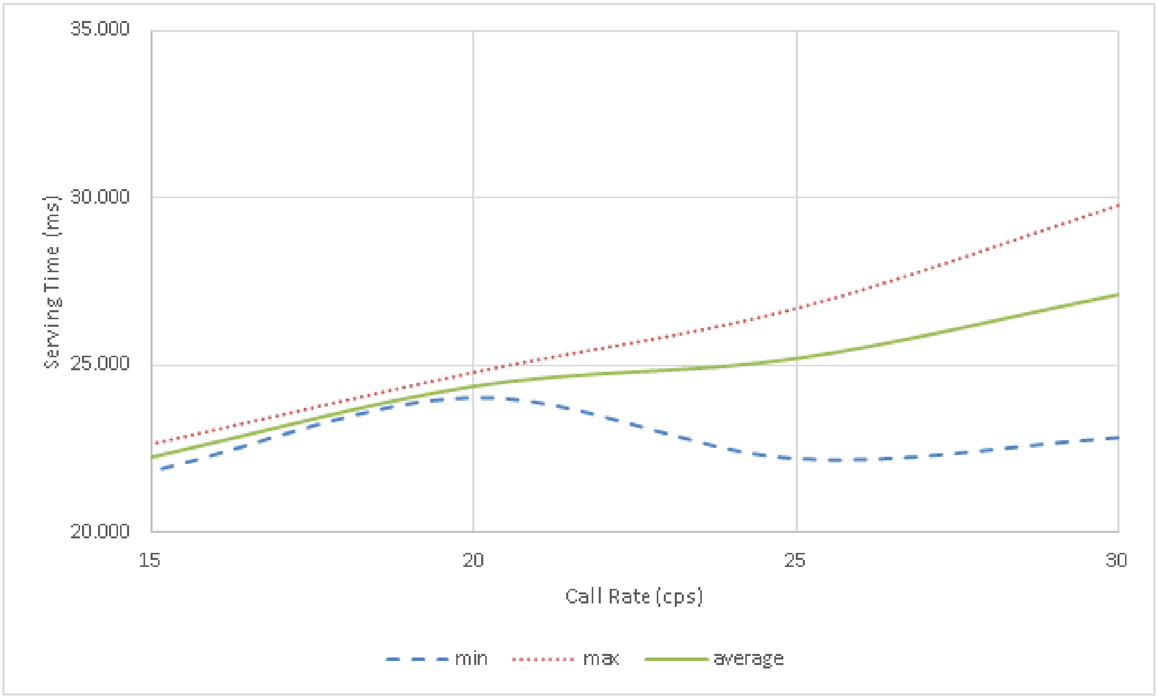


Figure 29. Serving Time at I-CSCF for Registration Procedure.

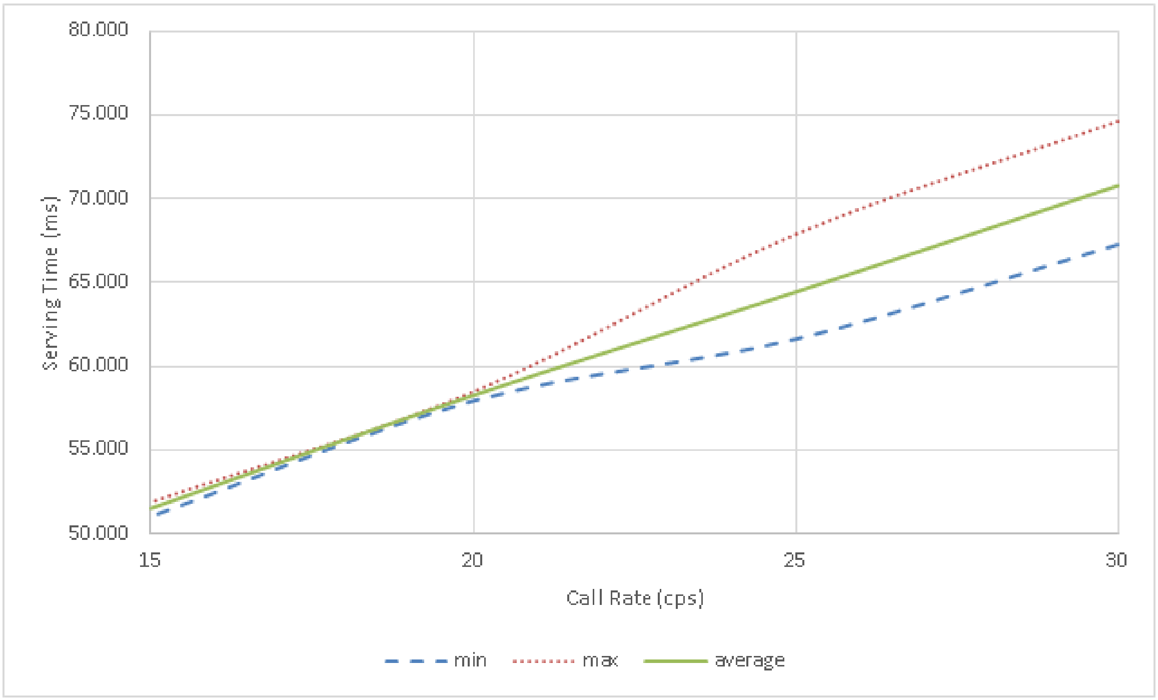


Figure 30. Serving Time at S-CSCF for Registration Procedure.

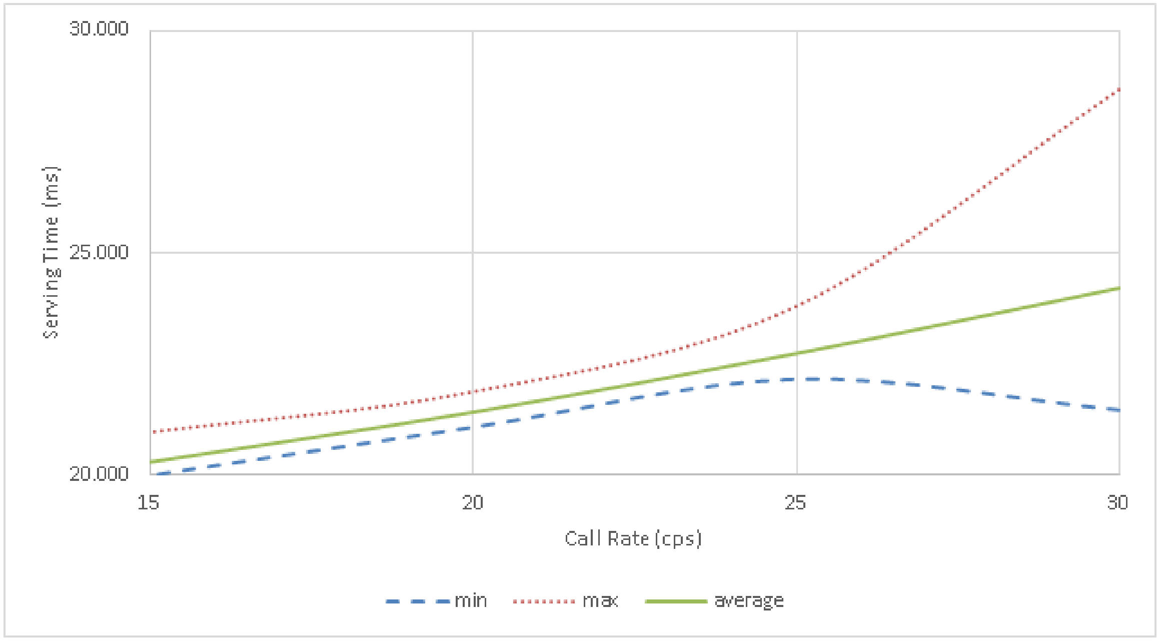


Figure 31. Serving Time at FHoSS for Registration Procedure.

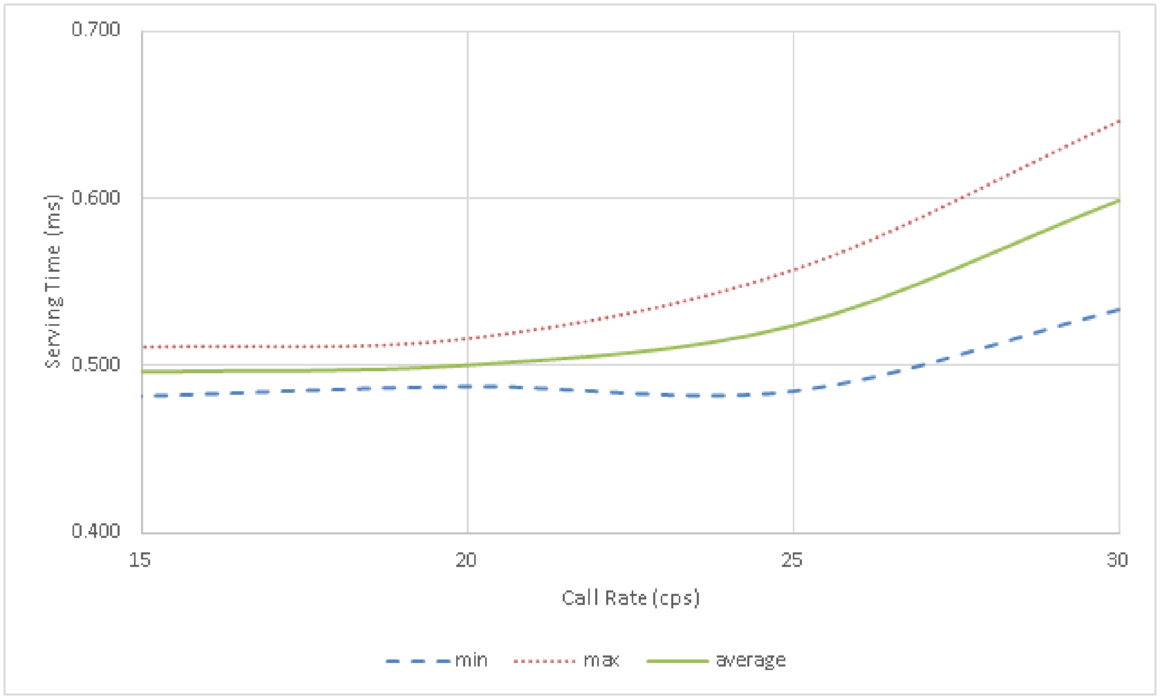


Figure 32. Serving Time of MySQL for Registration Procedure.

and calculated response time we will explain how the request serving time is collected in our test-bed system. As mentioned earlier, we measure the time based on Wireshark capture. However it cannot capture all transmitting packets. There are some packets missing. This

Call Rate	$\nu_{calculated}$	$\overline{x}_{measured}$	Difference
30cps	375.347	377.950	2.603
	401.804	403.720	1.916
	402.211	403.726	1.515
	438.693	439.715	1.022
	411.598	412.985	1.387
25cps	369.059	369.727	0.668
	373.206	373.992	0.786
	376.776	377.217	0.441
	367.927	368.571	0.644
	389.107	389.283	0.176
20cps	347.953	348.246	0.293
	346.623	347.339	0.716
	350.556	350.659	0.103
	349.361	349.978	0.617
	351.501	351.877	0.376
15cps	316.005	316.550	0.545
	314.397	314.756	0.359
	317.431	318.046	0.615
	317.006	317.368	0.362
	319.712	320.393	0.681

Table 10. Comparison between Measured and Calculated Response time (ms).

is the main cause for the difference. Let us define a packet flow as a procedure with a sent and a received packet. There are also some other conversational packets between these. In Wireshark capture, some procedures miss the expected sent or expected received packets for some subscribers. Or some procedures miss both of them and have only middle packets. These packet flow will be omitted from our calculation because there is no start time or stop time or both to calculate the time difference. Thus the measured and calculated responses do not match exactly. Furthermore, the system is not stable enough at 30cps, in which the retransmitted and drop packets are higher than other call rate. Therefore the difference between measured and calculated is wider.

4.6.2 Test Case 2: Registration, Call Setup and Termination Procedure

In the previous section, our test case has only registration procedure. We now run a test scenario that includes three procedures: registration, call set-up and call termination. According to the Equation (12), we need the arrival rate and the serving time at each node to calculate the sojourn time. We already calculated the arrival rate for registration procedure so in this section we will perform the calculation for the call set-up and call termination. Then we measure the average message processing time at each node. The total measured and calculated response time is compared at the end.

Arrival Rate for Call Setup Procedure

We classify the message types as in Table 11. The P-CSCF, I-CSCF and S-CSCF are classified as node number 1,2 and 3. From the Figure 43 in Appendix B, we derive the state flow for call setup procedure as in Figure 33.

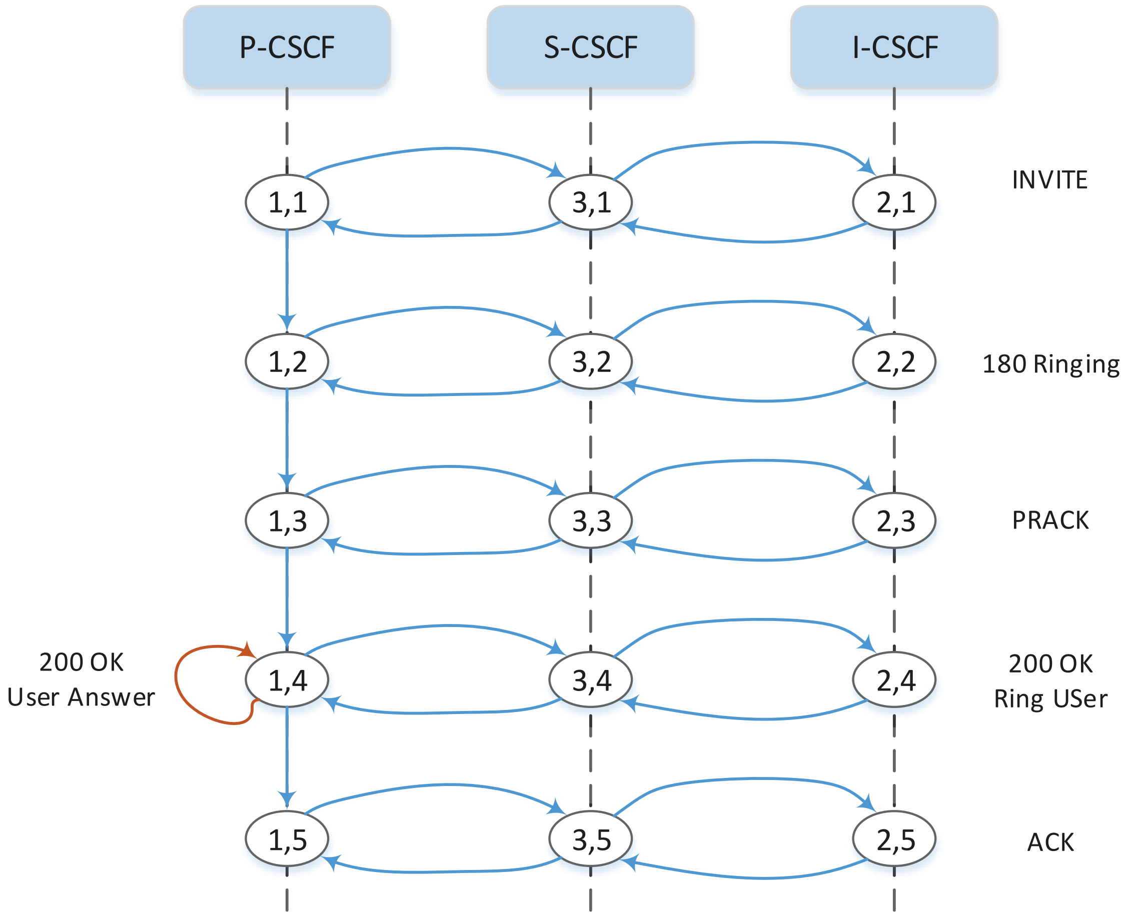


Figure 33. Call Setup Flow.

Message Type	***Number***
INVITE	1
180 Ringing	2
PRACK	3
200 OK	4
ACK	5

Table 11. Call Setup Message Type.

The transition probability matrix $\boldsymbol{\Theta}$ is

$$\boldsymbol{\Theta} = \begin{bmatrix} 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{21}$$

We get the transition rate matrix between the states as:

$$\boldsymbol{\lambda} = \begin{bmatrix} 60 & 60 & 30 & 60 & 60 & 30 & 60 & 60 & 30 & 90 & 60 & 30 & 60 & 60 & 30 \end{bmatrix}$$

We then can obtain the arrival rate for each node as:

$$\lambda_{P-CSCF} = 330 \; cps$$
$$\lambda_{I-CSCF} = 150 \; cps$$
$$\lambda_{S-CSCF} = 300 \; cps$$

Arrival Rate for Call Termination Procedure

From Figure 44 in Appendix B we derive that state flow for the termination process as in Figure 34. The BYE message is classified as type 1 and 200 OK messages as type 2.

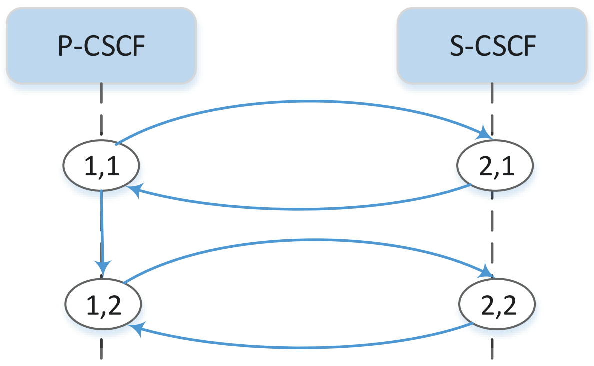


Figure 34. Call Termination Flow.

The transition probability matrix $\mathbf{\Theta}$ is

$$\mathbf{\Theta} = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{22}$$

We get the transition rate matrix between the states as:

$$\boldsymbol{\lambda} = \begin{bmatrix} 60 & 30 & 60 & 60 \end{bmatrix}$$

We then can obtain the arrival rate for each node as:

$$\lambda_{P-CSCF} = 120\ cps$$
$$\lambda_{S-CSCF} = 90\ cps$$

The average serving time at each node is shown in Table 12. These values have the same characteristics as values in test case 1. The confidence interval is wider at the higher rates than at the lower rates with the confidence of 98%. Using the arrival rates and the measured serving time we can calculate the sojourn time as in Equation (12) at each IMS node for each procedure. The calculated results for each procedure and the total calculated time are shown in Table 13. In Table 14, we compare the measured and calculated results. As we explained earlier, at the call rate of 30cps our test-bed is not stable, thus the differences are wider than at the lower rates. For lower rate, we can achieve the precision at around 1ms.

Call Rate	$\overline{x}_{P-CSCF}$	$\overline{x}_{I-CSCF}$	$\overline{x}_{S-CSCF}$	$\overline{x}_{FHoSS}$	$\overline{x}_{MySQL}$
30cps	51.341	27.931	78.614	26.230	0.694
	51.694	29.455	79.119	27.574	0.631
	51.585	29.799	80.453	27.099	0.673
	52.088	30.462	80.087	28.847	0.682
	50.102	31.837	81.630	27.002	0.705
25cps	49.484	25.730	76.318	24.850	0.612
	50.289	26.971	75.370	24.554	0.614
	49.867	27.545	76.511	25.276	0.656
	50.95	25.914	77.198	26.473	0.683
	50.637	26.514	77.567	25.002	0.675
20cps	47.216	23.791	73.317	22.635	0.593
	46.448	23.639	74.324	21.543	0.605
	46.947	21.799	72.532	20.269	0.582
	48.912	22.898	73.387	21.075	0.565
	48.214	21.236	74.861	20.274	0.568
15cps	41.742	20.064	70.615	19.935	0.523
	41.417	18.211	71.546	20.372	0.497
	39.628	17.707	70.417	21.241	0.569
	40.017	17.932	70.606	20.754	0.518
	40.151	18.547	69.938	19.775	0.553

Table 12. Average Request Serving Time (ms) for 3-procedure Test Case.

Call Rate	***Registration***	***Call Setup***	***Call Termination***	***Total Time***
30cps	449.793	863.478	181.271	1494.542
	461.331	873.198	182.482	1517.011
	462.997	881.580	183.598	1528.175
	472.270	884.391	184.238	1540.899
	468.175	885.858	181.809	1535.842
25cps	429.200	831.957	175.261	1436.418
	431.457	834.822	175.923	1442.202
	435.989	836.058	174.620	1446.667
	441.078	846.585	179.073	1466.736
	437.090	848.721	178.816	1464.627
20cps	403.947	794.526	167.724	1366.197
	399.625	795.504	167.195	1362.324
	386.377	782.226	166.401	1335.004
	398.504	802.443	171.186	1372.133
	395.252	812.253	174.644	1382.149
15cps	367.474	740.289	156.074	1263.837
	362.823	732.366	154.355	1249.544
	359.095	713.346	149.648	1222.089
	358.876	717.489	150.615	1226.980
	355.807	716.130	150.215	1222.152

Table 13. The Calculated Time (ms) for Each Procedure at IMS Nodes.

Call Rate	$\nu_{calculated}$	$\overline{x}_{measured}$	**Difference**
30cps	1494.542	1499.334	4.792
	1517.011	1519.449	2.438
	1528.175	1531.246	3.071
	1540.899	1543.685	2.786
	1535.842	1540.060	4.218
25cps	1436.418	1437.740	1.322
	1442.202	1443.412	1.210
	1446.667	1448.270	1.603
	1466.736	1468.311	1.575
	1464.627	1465.801	1.174
20cps	1366.197	1367.109	0.912
	1362.324	1363.357	1.033
	1335.004	1336.287	1.283
	1372.133	1373.268	1.135
	1382.149	1383.490	1.341
15cps	1263.837	1264.802	0.965
	1249.544	1250.828	1.284
	1222.089	1223.389	1.300
	1226.980	1228.302	1.322
	1222.152	1223.119	0.967

Table 14. Comparison between Measured and Calculated time (ms) for 3 Procedures.

In the previous papers, the performance evaluations were done mostly using various measuring tools. These papers focus on the standard functionalities of IMS components or only measure the delay and response time of the IMS system. There has not been any methodological model to evaluate the system. Recently in the paper [21], the author proposes a queueing model using M/M/1 queue but does not explain how to use it for the system evaluation. The author focuses on the selection algorithm for S-CSCF. Lately, in the paper [1], the author builds a test-bed system and collects the delay time. Then he derives trend lines for these data. There are no verifications for the work to see if it can adapt to a more complicated IMS system. Understanding these drawbacks, we propose a model that makes use of both experiment results and a queueing network to evaluate the response time for various scenarios. Our model takes the measured request serving time and the calculated arrival rate (using our mathematical formula) to evaluate the sojourn time at each node. With these results and the number of messages passing though each node, we can deduce the total response time for one or many procedures. Compared to a production network where there may be hundreds of nodes, our model is limited to a bounded number of nodes. We focus on the main four component types of an IMS: HSS, S-CSCF, P-CSCF and I-CSCF. Each of these component type is implemented on one node in our model. We can obtain an excellent correlation between the modelled results and the experimental results. Our model has been evaluated with two test cases. The first test case includes the registration procedure only. The second test case includes three procedures: registration, call setup and call termination. If the system implements more services and has more procedures, our model can adapt to these change as well. Furthermore, in our analysis we separate the FHoSS into two entities and manage to achieve the precise evaluation results. This proves that our model is flexible. Either the IMS components are separated or combined, our model can be contracted or extended to accommodate. In the two following sections, we introduce two new features of our model to evaluate the saturation point of the system and the CPU utilization of each IMS component. The saturation point of the system is the point over which the system becomes unstable. The CPU utilization is helpful when we can predict the CPU usage of each component and combine the components on the same machine to save resources.

4.7 System Performance Evaluation

In the previous section, we observe the system instability at the call rate of 30cps for an all-in-one IMS machine. Our question is: what is the maximum call rate for a specific system where all IMS components are running on different machines? This will be a critical system parameter when we consider the system capacity. Thus we will build a model to predict the system maximum call rate. In addition, the CPU usages for each IMS component are also derived so that we can combine several IMS components on one machine without wasting CPU resource.

4.7.1 System Saturation Point

In our queueing model, each of the IMS components is represented by a queue and a processing server. According to [25], the server utilization is the fraction of the time in which the server is occupied and is given by:

$$\rho = \frac{\lambda}{\mu} \tag{23}$$

where λ is the arrival rate and μ is the service rate. The condition for a stable system is $\rho < 1$. This means that the maximum arrival rate must be equal to or less than the service rate.

The arrival rate for each of the IMS components is given by [25]:

$$\lambda_i = \lambda_{0i} + \sum_{j=1}^{4} \lambda_j p_{ji} \quad i = 1...4 \tag{24}$$

where λ_{0i} is the external arrival rate to each IMS component and $\lambda_j p_{ji}$ is the transition probability between nodes. These probabilities are shown in Figure 35.

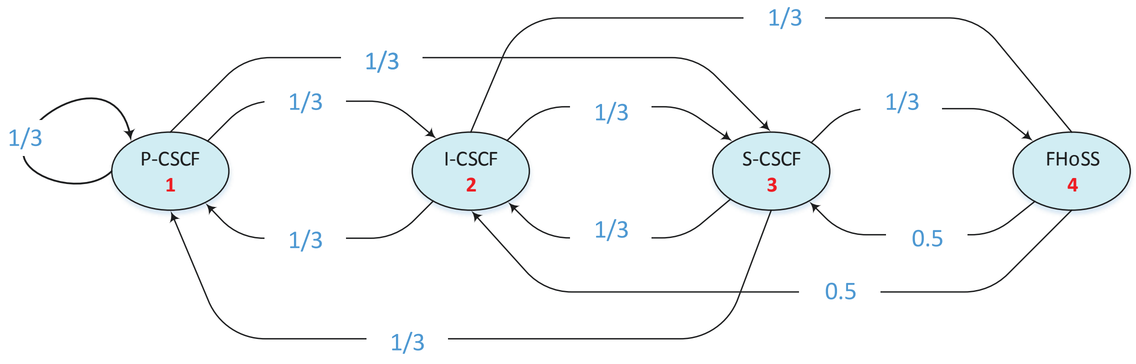


Figure 35. Transition Probability.

Thus we have four equations:

$$\begin{aligned}
\lambda_1 &= \lambda_{01} + \lambda_1 p_{11} + \lambda_2 p_{21} + \lambda_3 p_{31} + \lambda_4 p_{41} \\
\lambda_2 &= \lambda_{02} + \lambda_1 p_{12} + \lambda_2 p_{22} + \lambda_3 p_{32} + \lambda_4 p_{42} \\
\lambda_3 &= \lambda_{03} + \lambda_1 p_{13} + \lambda_2 p_{23} + \lambda_3 p_{33} + \lambda_4 p_{43} \\
\lambda_4 &= \lambda_{04} + \lambda_1 p_{14} + \lambda_2 p_{24} + \lambda_3 p_{34} + \lambda_4 p_{44}
\end{aligned}$$

Solving the system of equations we get the results:

$$\begin{aligned}
\lambda_1(PCSCF) &= 0.0013\lambda \\
\lambda_2(ICSCF) &= 0.00066\lambda \\
\lambda_3(SCSCF) &= 0.0023\lambda \\
\lambda_4(FHoSS) &= 0.00059\lambda
\end{aligned}$$

The service time of four IMS components are measured at the arrival rate from 5cps to 30cps. The results are shown in Table 15. We then use the Matlab *polyfit()* function to get the fitting curve for the data (see Appendix N). The intersection points between the service and arrival rate are the saturation points over which the system becomes unstable. Figure 36 and 37 show that S-CSCF has the lowest saturation point at around 30cps. If it is over 30cps S-CSCF is not stable, which leads to an unstable IMS system. This is an expected value that we already observed in the previous measurement.

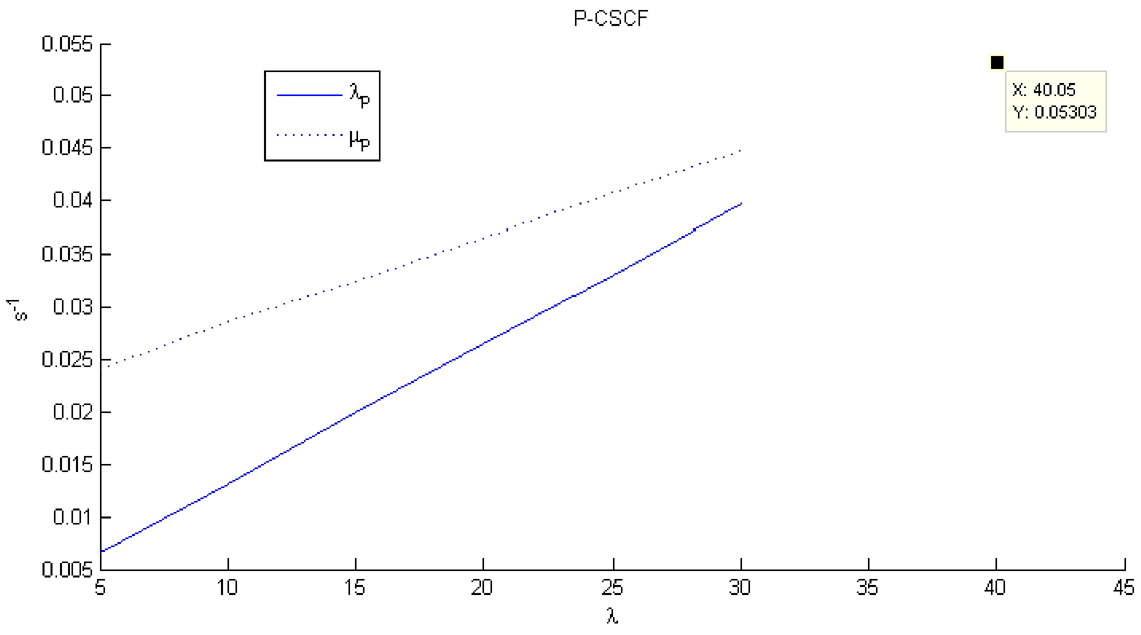
Rate	***P-CSCF***	***I-CSCF***	***S-CSCF***	***FHoSS***
5	0.0241	0.0167	0.0378	0.0146
10	0.0286	0.0196	0.0437	0.0177
15	0.0324	0.0221	0.0517	0.0200
20	0.0365	0.0243	0.0584	0.0219
25	0.0408	0.0265	0.0642	0.0236
30	0.0447	0.0287	0.0689	0.0254

Table 15. Service Time of Four IMS Components in All-in-One Machine.

We will verify our model against an IMS system where all four components are separated in four different physical machines. The measured service time is shown in Table 16. Figure 38 and 39 show that S-CSCF has the lowest saturation point at around 91cps. When we run the simulation with over 91cps, the system is not stable and the IHS rate increases over the acceptance limit, which causes the simulation to stop (Figure 40).

Rate	***P-CSCF***	***I-CSCF***	***S-CSCF***	***FHoSS***
5	0.0134	0.0078	0.0203	0.0069
10	0.0193	0.0109	0.0300	0.0097
15	0.0261	0.0135	0.0413	0.0127
20	0.0328	0.0171	0.0532	0.0156
25	0.0387	0.0206	0.0641	0.0183
30	0.0448	0.023	0.0749	0.0213

Table 16. Service Time of Four IMS Components in 4 Machines.



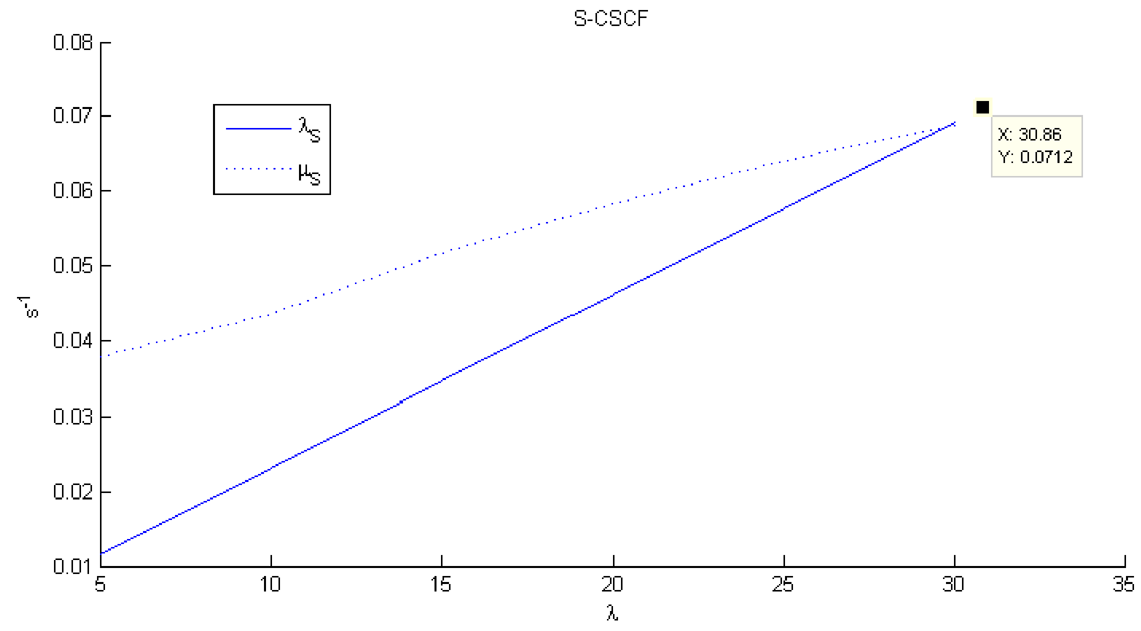
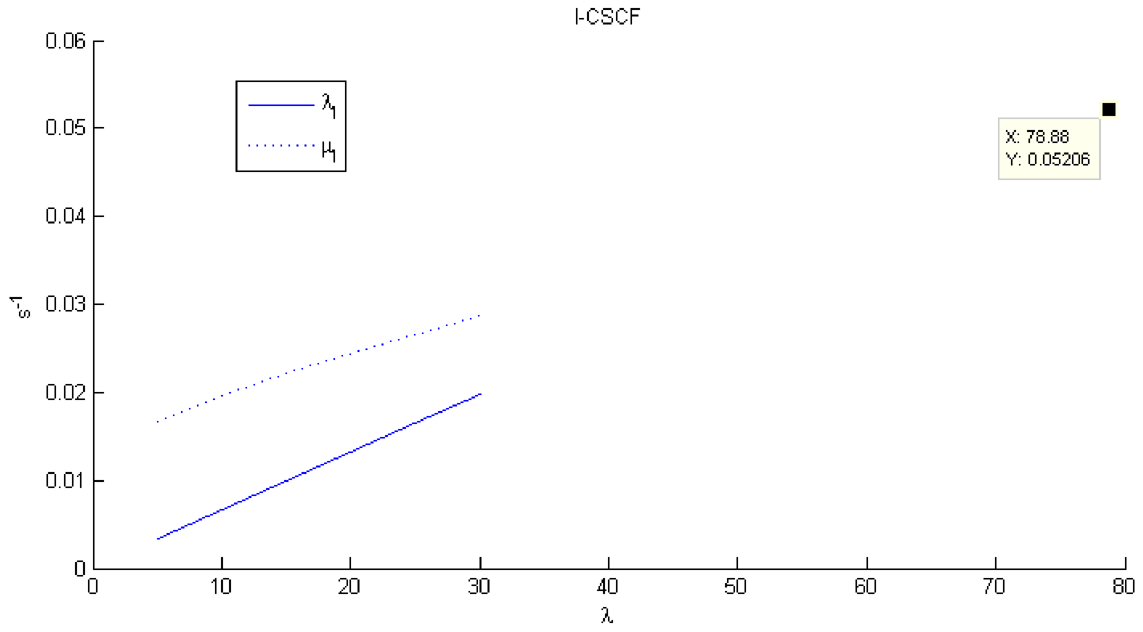


Figure 36. Saturation Points of P-CSCF and S-CSCF in All-in-One Machine.



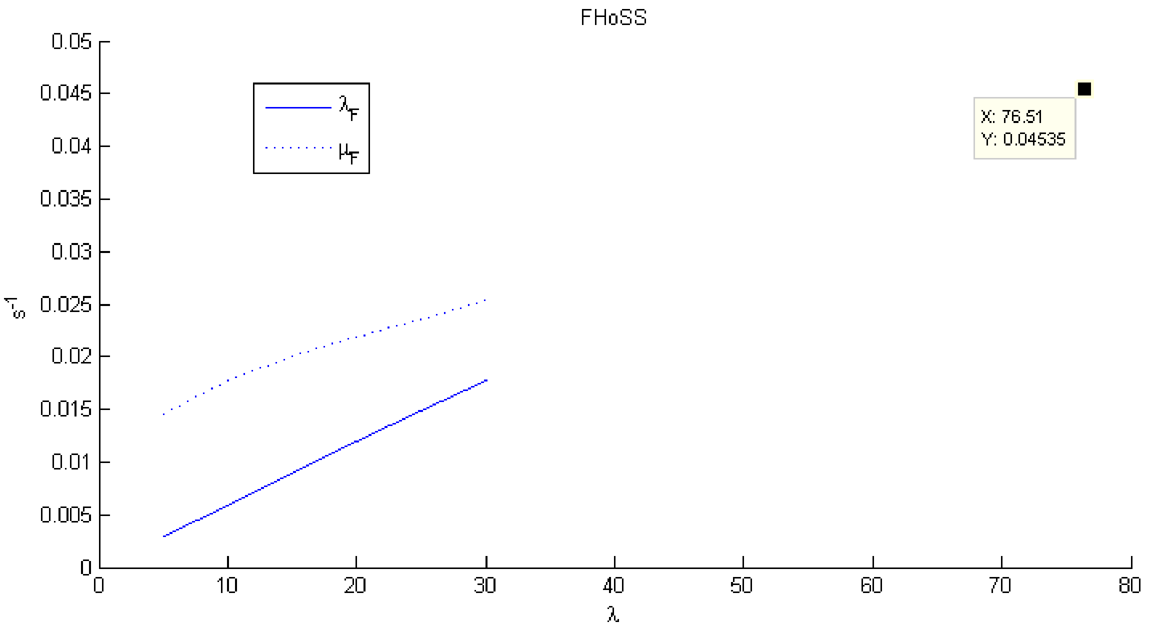
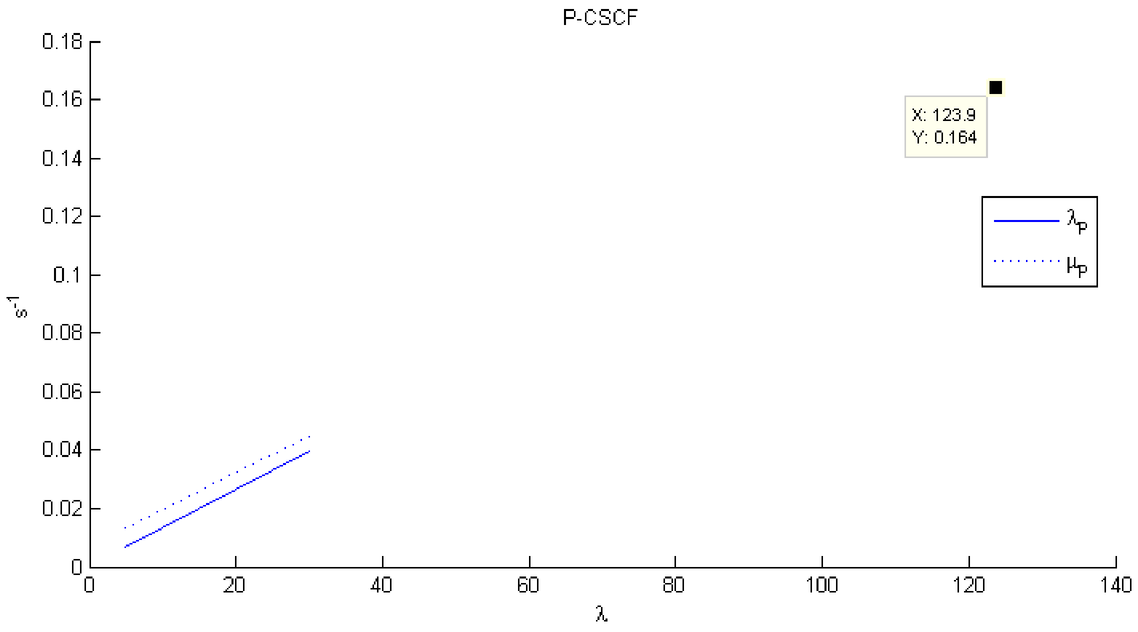


Figure 37. Saturation Points of I-CSCF and FHoSS in All-in-One Machine.



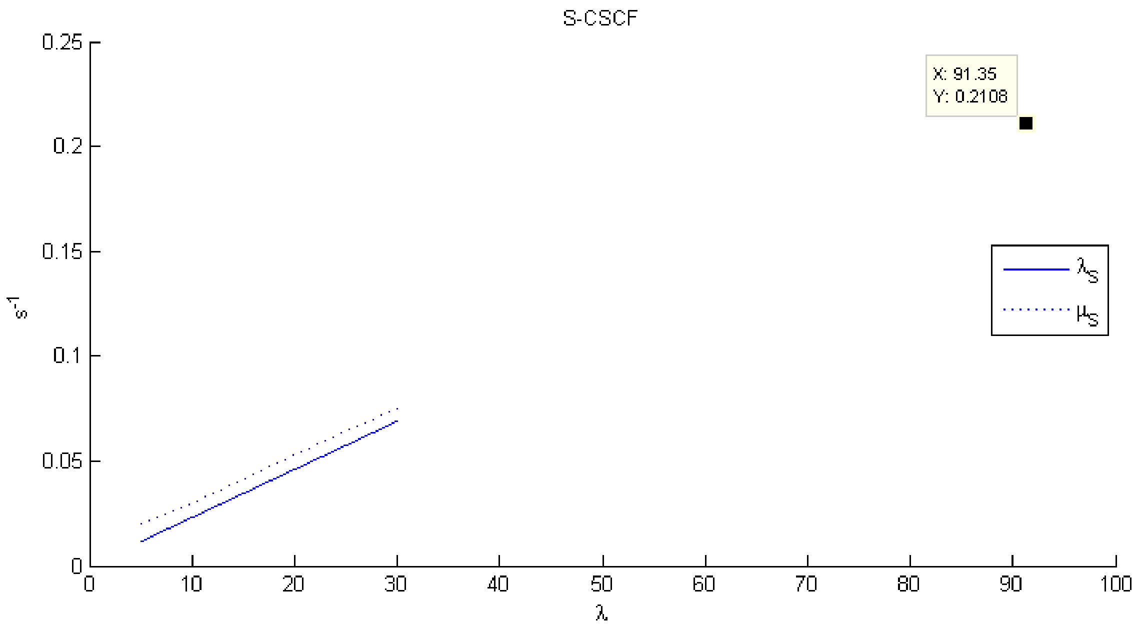
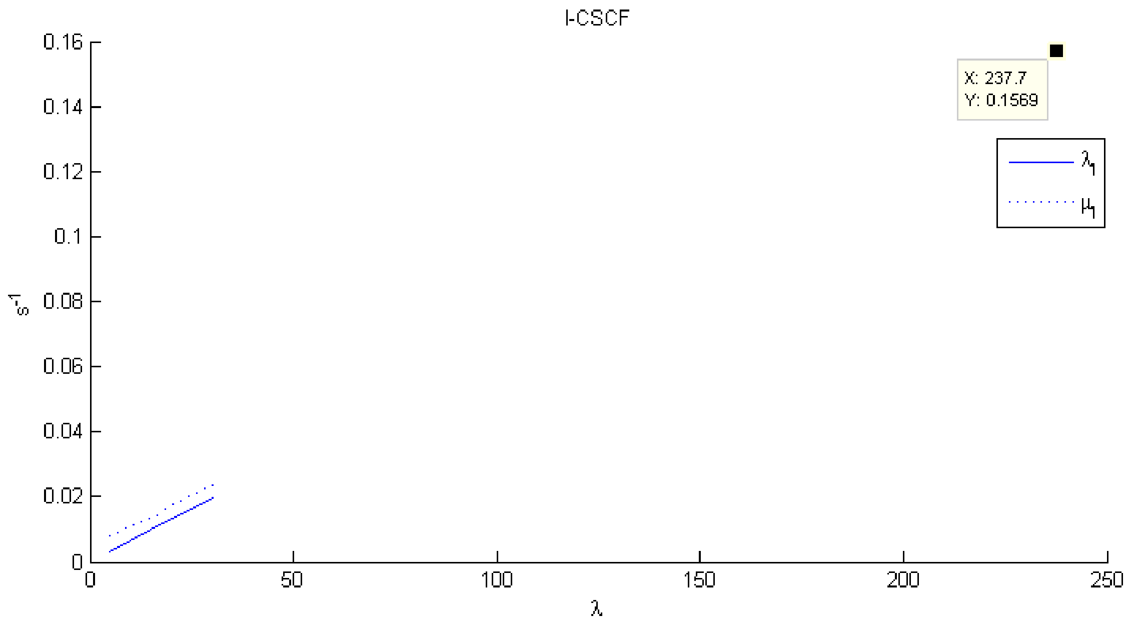


Figure 38. Saturation Points of P-CSCF and S-CSCF in 4 Machines.



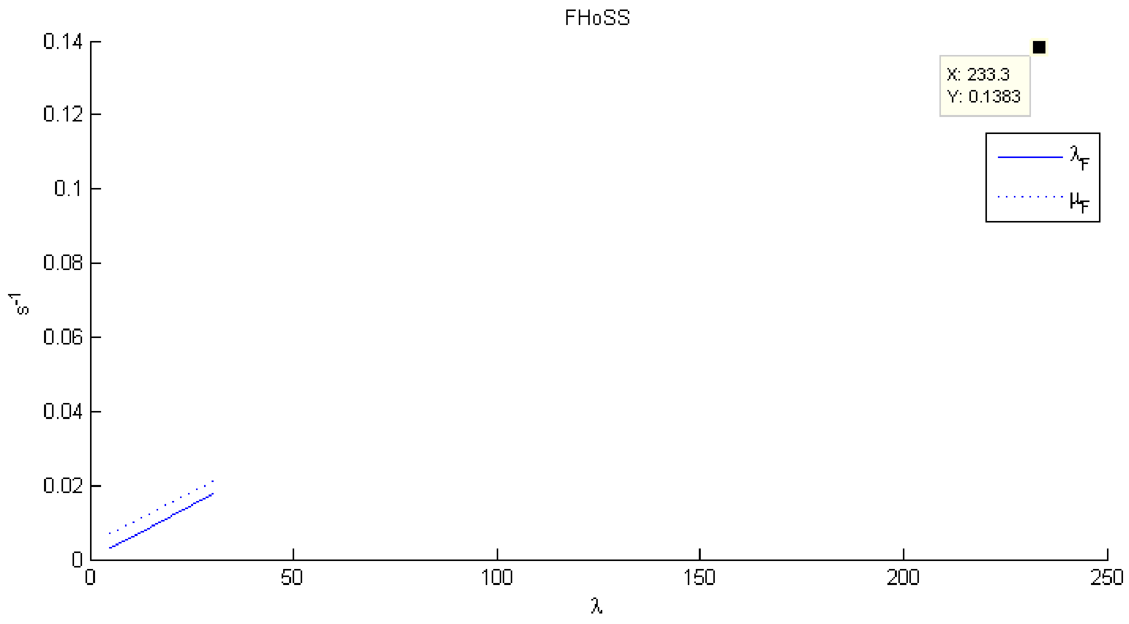
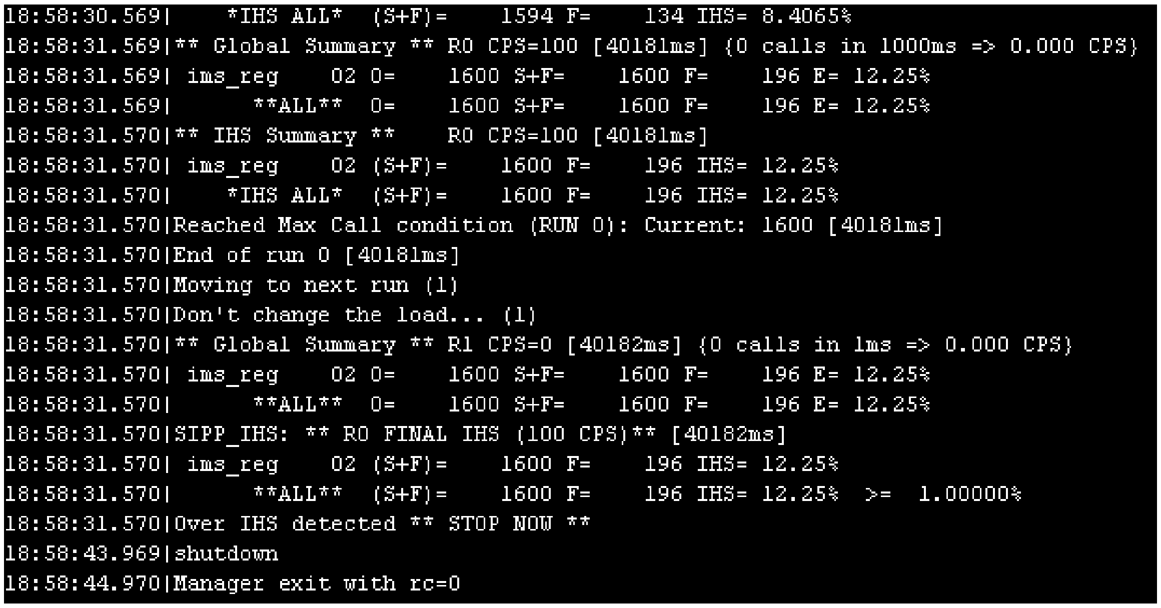


Figure 39. Saturation Points of I-CSCF and FHoSS in 4 Machines.



```
18:58:30.569|    *IHS ALL*  (S+F)=    1594 F=    134 IHS= 8.4065%
18:58:31.569|** Global Summary ** R0 CPS=100 [40181ms] (0 calls in 1000ms => 0.000 CPS)
18:58:31.569| ims_reg    02 O=    1600 S+F=    1600 F=    196 E= 12.25%
18:58:31.569|      **ALL**  O=    1600 S+F=    1600 F=    196 E= 12.25%
18:58:31.570|** IHS Summary **    R0 CPS=100 [40181ms]
18:58:31.570| ims_reg    02 (S+F)=    1600 F=    196 IHS= 12.25%
18:58:31.570|    *IHS ALL*  (S+F)=    1600 F=    196 IHS= 12.25%
18:58:31.570|Reached Max Call condition (RUN 0): Current: 1600 [40181ms]
18:58:31.570|End of run 0 [40181ms]
18:58:31.570|Moving to next run (1)
18:58:31.570|Don't change the load... (1)
18:58:31.570|** Global Summary ** R1 CPS=0 [40182ms] (0 calls in 1ms => 0.000 CPS)
18:58:31.570| ims_reg    02 O=    1600 S+F=    1600 F=    196 E= 12.25%
18:58:31.570|      **ALL**  O=    1600 S+F=    1600 F=    196 E= 12.25%
18:58:31.570|SIPP_IHS: ** R0 FINAL IHS (100 CPS)** [40182ms]
18:58:31.570| ims_reg    02 (S+F)=    1600 F=    196 IHS= 12.25%
18:58:31.570|      **ALL**  (S+F)=    1600 F=    196 IHS= 12.25%  >=  1.00000%
18:58:31.570|Over IHS detected ** STOP NOW **
18:58:43.969|shutdown
18:58:44.970|Manager exit with rc=0
```

Figure 40. Unstable System When The Call Rate Is Over The Saturation Point.

4.7.2 System CPU Utilization

In this section, we will explore the CPU utilization for each IMS component. In our test-bed, we implement the three CSCFs and FHoSS on four separate machines with identical CPU. We fed into the system with the call rate from 5cps to 40cps and measured the CPU usage. The results are shown in Table 17.

Rate	***P-CSCF***	***I-CSCF***	***S-CSCF***	***FHoSS***
5	4.98	5.67	3.24	2.66
10	9.87	11.23	6.58	5.62
15	14.76	16.38	9.72	8.28
20	19.94	22.69	12.76	11.44
25	24.73	27.83	16.60	13.80
30	29.13	33.36	19.94	16.46
35	34.30	38.46	22.68	19.32
40	39.84	44.92	25.72	22.38

Table 17. CPU Utilization of 4 IMS Components.

We then use the Matlab *polyfit()* function to get the fitting curve for the CPU utilization (see Appendix O and Figure 41). We can get their functions as:

$$\begin{aligned} U_{P-CSCF} &= 0.9859\lambda \\ U_{I-CSCF} &= 0.6485\lambda \\ U_{S-CSCF} &= 1.1119\lambda \\ U_{FHoSS} &= 0.5558\lambda \end{aligned}$$

At the call rate of 90cps, we have the CPU utilization at each IMS component as:

$$\begin{aligned} U_{P-CSCF} &= 88.73\% \\ U_{I-CSCF} &= 58.37\% \\ U_{S-CSCF} &= 100.07\% \\ U_{FHoSS} &= 50.02\% \end{aligned}$$

S-CSCF has the highest load following by P-CSCF. This reflects exactly the saturation point that we discovered in the previous section. At the call rate of 30cps, we have

$$\begin{aligned} U_{P-CSCF} &= 29.58\% \\ U_{I-CSCF} &= 19.46\% \\ U_{S-CSCF} &= 33.36\% \\ U_{FHoSS} &= 16.67\% \end{aligned}$$

This is the case of all-on-one machine in which we have the total CPU utilization of nearly 100%. By manipulating the above equations, we can combine the IMS components so that we can efficiently use the CPU resource. For example, at the call rate of 80cps, the CPU utilization of I-CSCF and FHoSS are nearly 50%. Thus we can integrate them on one machine and P-CSCF and S-CSCF on other two machines.

In this chapter, we have introduced our test-bed environment which is installed on Ubuntu using the Open Source IMS Core software. The test-bed can be implemented on one machine or multiple machines. We then used the SIPp generator to inject traffic into the IMS system. The response time is measured at each IMS node. The obtained functions in Chapter 3 are used to calculate the total response time of registration, call setup and call termination procedure. We compared these calculated results with the measured results. We can obtain around millisecond precision. In addition, we also used our model to calculate the system saturation point and the CPU usage of each IMS component.

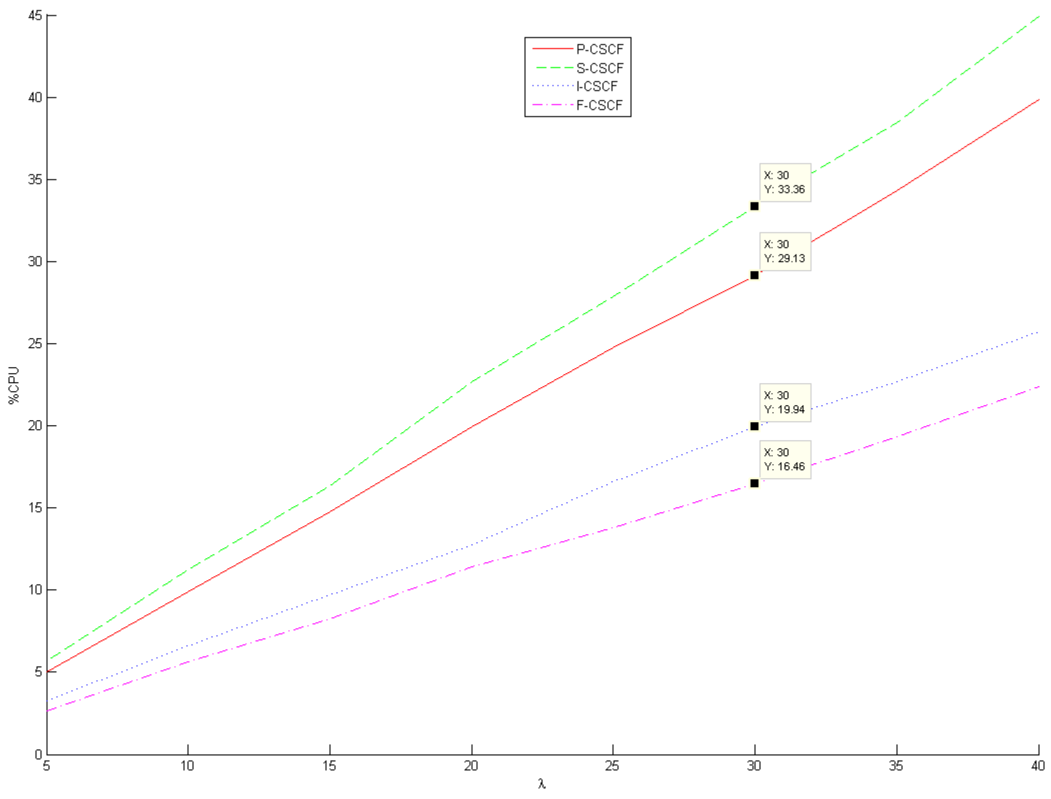


Figure 41. CPU Utilization Curves.

Chapter 5

Conclusion and Future Work

The importance of the IP Multimedia Subsystem increases as more and more service providers integrate such systems into their networks recently. The IMS performance is the main concern when the number of subscribers dramatically multiplies. This thesis explores a methodological model to effectively evaluate the performance of IMS. The model covers all the main IMS components and is expandable to include new nodes as the modelled network grows.

After presenting the fundamental aspects of four IMS components (P-CSCF, I-CSCF, S-CSCF and HSS) and basic queueing concepts, we build up our model as a queueing network in which each transmitted message is considered as a request. Potential components of a request are modelled as classes. The requests travel from one queue to another with or without changing classes. Our model has precisely classified requests into many different classes so that all the transition states are included. In addition, we also form a transition probability matrix to describe the characteristics of request flows inside the IMS system. With a specified IMS call rate, we easily obtain the arrival rate for every node in the network. Our model is not simply a mathematical equation with predefined constants. We measure the average response time of every node to provide a reliable evaluation result. We verify the proposed model with the Open Source IMS Core System. We obtain around millisecond precision in our test-cases for registration, call setup and termination procedures. The model expandability is verified when we break down the physical HSS into two logical components: one to process requests and another to query the subscriber database. Using this concept, we also add multiple nodes into an existing network evaluation without losing precision. Our model also finds the saturation point and the CPU utilization of an IMS system. The saturation point is the maximum limit over which the system becomes unstable and unpredictable. This limit helps us to prepare for an increasing number of subscribers. We also calculate the CPU usage of each IMS component and combine them in one machine so that the CPU resource can be used efficiently. This is really useful in a production network where the server resources are really expensive.

The proposed model has achieved our goal to effectively evaluate the IMS performance. Because of its scalability, this model can be applied to a live production network to study the request flow behaviours and thus deduce the system performance. Furthermore, when the service providers gradually use virtualization or cloud technologies for an IMS system, our model can adapt to the new environment as well. An extension to our work can be done with the request priority. In the real-time IMS network, the service providers might give processing priority for certain request types. Our model can adapt to this requirement as we already had the requests classified. A modification to our evaluation model is also possible.

Appendix

A Scripts to Start IMS Components

For the ease of provisioning, we created two scripts to start IMS (one script without logging and another script with logging.) The content of the scripts are shown below.

- ***startIMS.sh*** script is used to start all IMS components including xCSCF and FHoSS in separate terminal tabs (Figure 42).

```bash
#!/bin/bash
# This Program start all components of IMS including xCSCF and FHoSS.
# Each component runs on a separate terminal tab.

clear
read -p "About to launch IMS Component. Press Enter to continue... "

# Set DNS resolver
cp /etc/resolv.conf.ims /etc/resolv.conf
cp /etc/hosts.ims /etc/hosts

# Set $JAVA_HOME
cd /opt/OpenIMSCore
echo -e "\n\nSetting JAVA_HOME location:"
echo -e "\tCurrent: $JAVA_HOME"
_JLocation=$(readlink -f /usr/bin/java | sed "s:jre/bin/java::")
export JAVA_HOME=$_JLocation
echo -e "\tSet to: $JAVA_HOME"

# Launch IMS components in separate terminal tabs
_cmd_P="./pcscf.sh"
_cmd_I="./icscf.sh"
_cmd_S="./scscf.sh"
_cmd_H="cd FHoSS/deploy/;./startup.sh"

echo -e "\n\nIMS is running... "
```

```
gnome-terminal --tab -t PCSCF -e $_cmd_P --tab -t ICSCF -e $_cmd_I --tab -t
     SCSCF -e $_cmd_S --tab -t FHoSS -e "bash -c '$_cmd_H';bash"
```

- ***startIMS_log.sh*** script is the same as ***startIMS.sh*** but with logging capability.

```
#!/bin/bash
# This Program start all components of IMS including xCSCF and FHoSS.
# Each component runs on a seperate terminal tab. Logging is enabled
# and save in ./logs/

clear
read -p "About to launch IMS Component. Press Enter to continue... "

# Set DNS resolver
cp /etc/resolv.conf.ims /etc/resolv.conf
cp /etc/hosts.ims /etc/hosts

# Set $JAVA_HOME
cd /opt/OpenIMSCore
echo -e "\n\nSetting JAVA_HOME location:"
echo -e "\tCurrent: $JAVA_HOME"
_JLocation=$(readlink -f /usr/bin/java | sed "s:jre/bin/java::")
export JAVA_HOME=$_JLocation
echo -e "\tSet to: $JAVA_HOME"

# Launch IMS components in seperate terminal tabs.
echo -e "\n\nIMS is running... "
gnome-terminal --tab -t PCSCF -e 'bash -c "./pcscf.sh 2>&1 | tee ./logs/
    pcscf.log.tmp"' --tab -t ICSCF -e 'bash -c "./icscf.sh 2>&1 | tee ./
    logs/icscf.log.tmp"' --tab -t SCSCF -e 'bash -c "./scscf.sh 2>&1 | tee
    ./logs/scscf.log.tmp"' --tab -t FHoSS -e 'bash -c "cd FHoSS/deploy/;./
    startup.sh"'

# Remove color code for easy reading.
cd /opt/OpenIMSCore/logs
cat pcscf.log.tmp | sed -r "s/\x1B\[([0-9]{1,3}((;[0-9]{1,3})*)?)?[m|K]//g"
     > pcscf.log
rm pcscf.log.tmp
cat icscf.log.tmp | sed -r "s/\x1B\[([0-9]{1,3}((;[0-9]{1,3})*)?)?[m|K]//g"
     > icscf.log
rm icscf.log.tmp
cat scscf.log.tmp | sed -r "s/\x1B\[([0-9]{1,3}((;[0-9]{1,3})*)?)?[m|K]//g"
     > scscf.log
rm scscf.log.tmp
```

How to use the sripts?

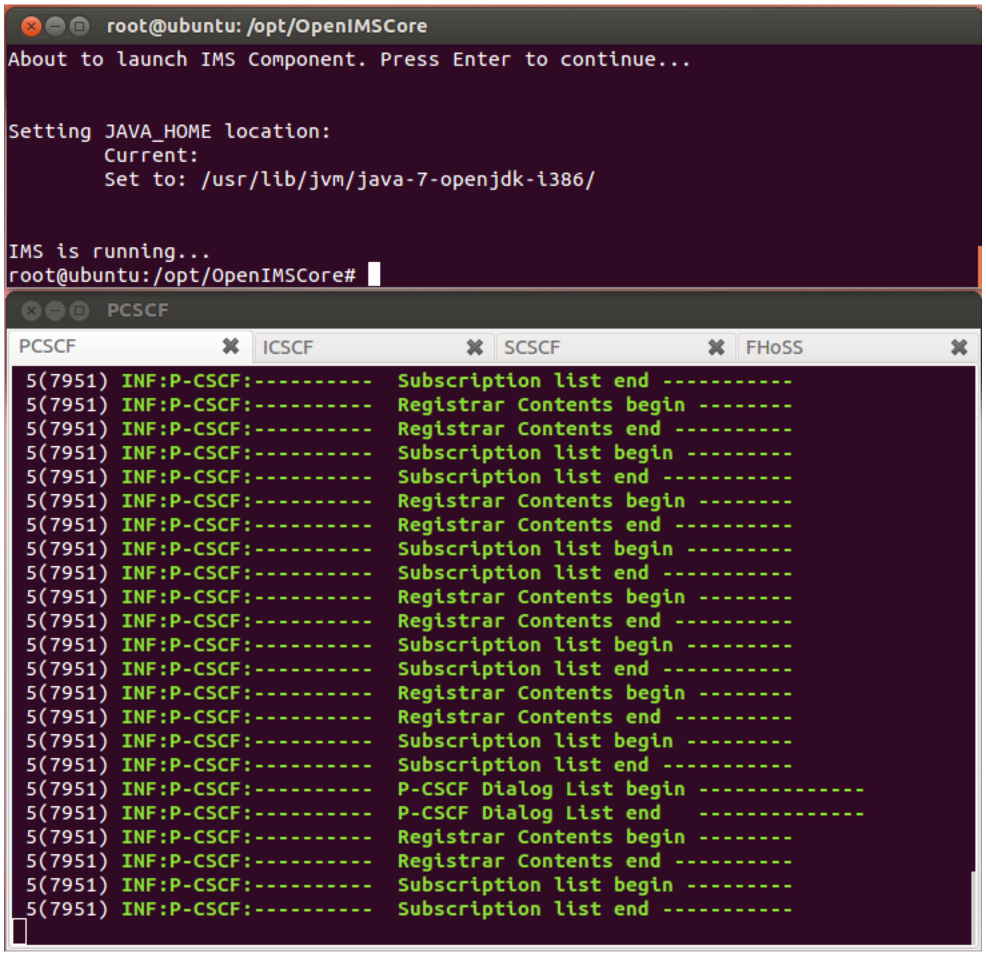
Copy the scripts to ***/opt/OpenIMSCore*** then execute:

```
user@ubuntu# cd /opt/OpenIMSCore/
user@ubuntu# sudo ./startIMS.sh
```

This will start four IMS components (PCSCF, ICSCF, SCSCF and HSS) in 4 separate terminal tabs. If we want to log the output of each session, use the other script:

```
user@ubuntu# cd /opt/OpenIMSCore/
user@ubuntu# sudo ./startIMS_log.sh
```

This script records the terminal tabs' output of P-CSCF, I-CSCF and S-CSCF. These logs are stored in ***/opt/OpenIMSCore/logs***. Log of FHoSS is located in ***/opt/OpenIMSCore/FHoSS/deploy/logs***.



```
About to launch IMS Component. Press Enter to continue...


Setting JAVA_HOME location:
        Current:
        Set to: /usr/lib/jvm/java-7-openjdk-i386/


IMS is running...
root@ubuntu:/opt/OpenIMSCore# 
```

```
 5(7951) INF:P-CSCF:----------  Subscription list end -----------
 5(7951) INF:P-CSCF:----------  Registrar Contents begin --------
 5(7951) INF:P-CSCF:----------  Registrar Contents end ----------
 5(7951) INF:P-CSCF:----------  Subscription list begin ---------
 5(7951) INF:P-CSCF:----------  Subscription list end -----------
 5(7951) INF:P-CSCF:----------  Registrar Contents begin --------
 5(7951) INF:P-CSCF:----------  Registrar Contents end ----------
 5(7951) INF:P-CSCF:----------  Subscription list begin ---------
 5(7951) INF:P-CSCF:----------  Subscription list end -----------
 5(7951) INF:P-CSCF:----------  Registrar Contents begin --------
 5(7951) INF:P-CSCF:----------  Registrar Contents end ----------
 5(7951) INF:P-CSCF:----------  Subscription list begin ---------
 5(7951) INF:P-CSCF:----------  Subscription list end -----------
 5(7951) INF:P-CSCF:----------  Registrar Contents begin --------
 5(7951) INF:P-CSCF:----------  Registrar Contents end ----------
 5(7951) INF:P-CSCF:----------  Subscription list begin ---------
 5(7951) INF:P-CSCF:----------  Subscription list end -----------
 5(7951) INF:P-CSCF:----------  P-CSCF Dialog List begin --------------
 5(7951) INF:P-CSCF:----------  P-CSCF Dialog List end   --------------
 5(7951) INF:P-CSCF:----------  Registrar Contents begin --------
 5(7951) INF:P-CSCF:----------  Registrar Contents end ----------
 5(7951) INF:P-CSCF:----------  Subscription list begin ---------
 5(7951) INF:P-CSCF:----------  Subscription list end -----------
```

Figure 42. Start IMS Components With Scripts.

B Call Setup and Termination Procedure

The call setup procedure between two user equipments is shown below:

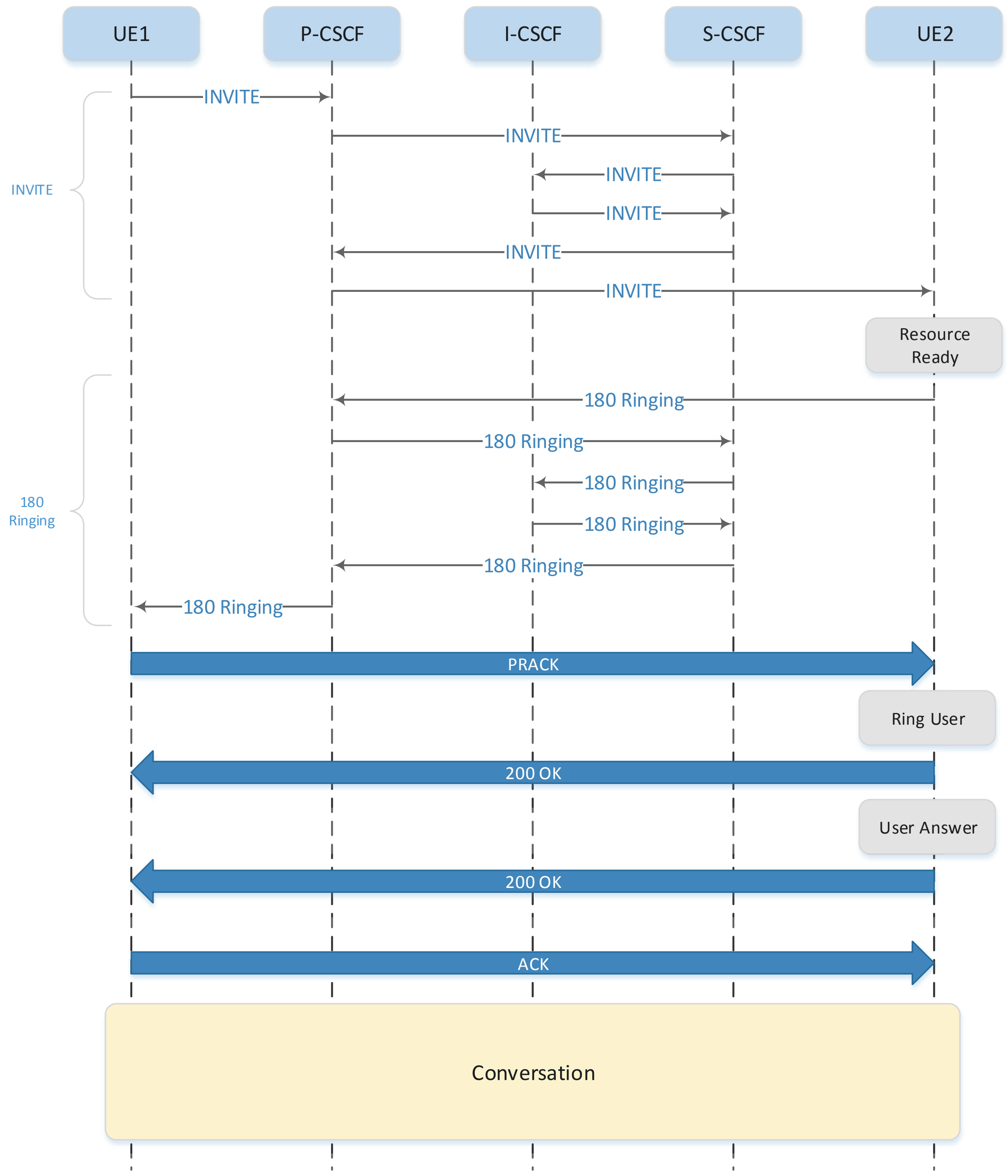


Figure 43. Call Setup Procedure.

The call termination procedure is shown below:

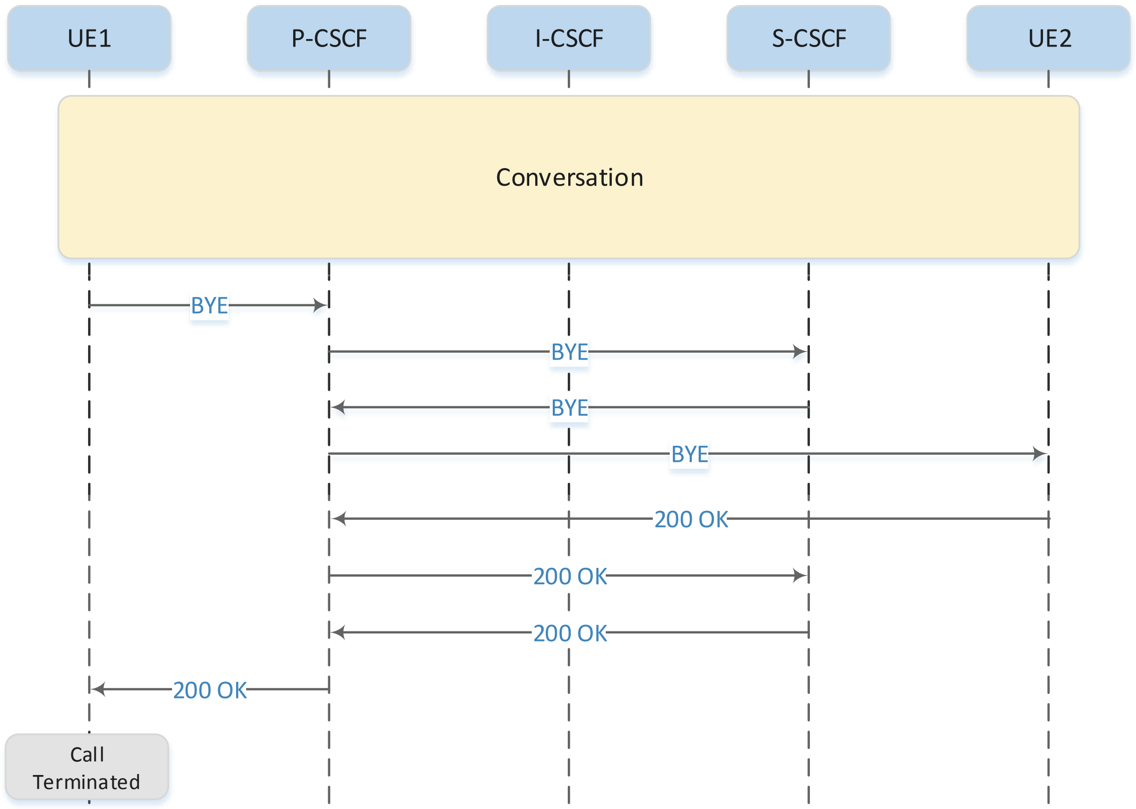


Figure 44. Call Termination Procedure.

C Open Source IMS Implementation Steps

C.1 Prerequisite Packages

In order to compile the IMS source codes and run application for core system, we need to have the following packages installed on Ubuntu:

- ***subversion*** - To download the IMS source codes.

  ```
  user@ubuntu# sudo apt-get install subversion
  ```

- ***gcc, make*** - To compile CSCFs source codes.

  ```
  user@ubuntu# sudo apt-get install gcc
  user@ubuntu# sudo apt-get install make
  ```

- ***openjdk-7-jdk, openjdk-7-jre-headless, openjdk-7-jre-lib*** - To run HSS.

  ```
  user@ubuntu# sudo apt-get install openjdk-7-jdk
  user@ubuntu# sudo apt-get install openjdk-7-jre-headless
  user@ubuntu# sudo apt-get install openjdk-7-jre-lib
  ```

- ***ant*** - To compile HSS source code

  ```
  user@ubuntu# sudo apt-get install ant
  ```

- ***mysql-server-5.5*** - MySQL database to store IMS subscription information. During installation it will ask for database root password. Choose an appropriate password.

  ```
  user@ubuntu# sudo apt-get install mysql-server-5.5
  ```

- ***mysql-client-5.5*** - MySQL client to retrieve IMS subscriptions.

  ```
  user@ubuntu# sudo apt-get install mysql-client-5.5
  ```

- ***libmysqlclient-dev*** - MySQL development libraries and header files.

  ```
  user@ubuntu# sudo apt-get install libmysqlclient-dev
  ```

- ***bison*** - C parser for CSCF source code.

  ```
  user@ubuntu# sudo apt-get install bison
  ```

- ***flex***

```
user@ubuntu# sudo apt-get install flex
```

- ***libxml2-dev***

  ```
  user@ubuntu# sudo apt-get install libxml2-dev
  ```

- ***ipsec-tools*** - For IP Security Association

  ```
  user@ubuntu# sudo apt-get install ipsec-tools
  ```

- ***curl***

  ```
  user@ubuntu# sudo apt-get install curl
  ```

- ***libcurl4-gnutls-dev***

  ```
  user@ubuntu# sudo apt-get install libcurl4-gnutls-dev
  ```

- ***bind9*** - DNS server to resolve IMS domain.

  ```
  user@ubuntu# sudo apt-get install bind9
  ```

While most of us prefer to run one command e.g. "*sudo apt-get install subversion gcc make ...*" to install all packages in one attempt, we suggest to install packages one by one so that any errors can be spotted and corrected.

C.2 OpenIMS Core Source Code

As we mentioned in the introduction, the IMS CSCF components are built upon open source SER software. The author implemented them using C programming language. The source codes are hosted on Berlin Open Source (Berlios) website. It is a project created by Fraunhofer Institute from Berlin to coordinate different open source softwares. In addition to the CSCF, FOKUS developed his own prototype of HSS. It is called FOKUS Home Subscriber Server (FHoSS) which is completely written in Java. From this point of view, we can separate source codes of CSCF and HSS to implement on different servers. We can also provision each type of CSCF (P, I, S) on one or many servers. The following steps show how to download the IMS source code on Ubuntu.

- Create "/opt/OpenIMSCore" and go inside the directory:

```
user@ubuntu# mkdir /opt/OpenIMSCore
user@ubuntu# cd /opt/OpenIMSCore
```

- Create "ser_ims" directory and get SER source code:

```
user@ubuntu# mkdir ser_ims
user@ubuntu# svn checkout http://svn.berlios.de/svnroot/repos/
    openimscore/ser_ims/trunk ser_ims
```

- Create "FHoSS" directory and get HSS source code:

```
user@ubuntu# mkdir FHoSS
user@ubuntu# svn checkout http://svn.berlios.de/svnroot/repos/
    openimscore/FHoSS/trunk FHoSS
```

C.3 Configure DNS Server

DNS plays an important role for IMS system. It helps resolves the home or visit domain or subscribers. In IMS core system, we can point UEs directly to the IP of CSCFs or HSS but it is not scalable when the number of components increases. Most small test-beds use this approach. Even though the DNS server implementation is complex and requires some extensive knowledges, we prefer to use DNS for scalability. The most common DNS server is BIND9. In the following steps we will setup DNS server for the domain ***open-ims.test*** on the loop-back interface ***127.0.0.1***. Because we install all components on one workstation, all of them will have the same IP address but different port numbers.

Configure DNS Client Side

The Client Side represents the CSCFs and HSS where the DNS queries originating. It includes ***/etc/hosts*** and ***/etc/resolv.conf*** file.

- Configure ***/etc/hosts*** file

```
user@ubuntu# sudo gedit /etc/hosts

127.0.0.1 localhost
127.0.0.1 localhost.open-ims.test localhost
127.0.0.1 hss.open-ims.test hss
127.0.0.1 pcscf.open-ims.test pcscf
127.0.0.1 scscf.open-ims.test scscf
127.0.0.1 icscf.open-ims.test icscf
```

- Configure the resolver ***/etc/resolv.conf***. This file is controlled by the host resolver and is reset when the workstation restarts.

```
user@ubuntu# sudo gedit /etc/resolv.conf

nameserver 127.0.0.1
domain open-ims.test
search open-ims.test
```

Configure DNS Server Side

- Configure ***/etc/bind/named.conf***

```
user@ubuntu# sudo gedit /etc/bind/named.conf

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

- Configure ***/etc/bind/named.conf.options***

```
user@ubuntu# sudo gedit /etc/bind/named.conf.options

options {
  directory "/var/cache/bind";

  forwarders {
    127.0.0.1;
  };

  dnssec-validation auto;

  auth-nxdomain no;    # conform to RFC1035
  listen-on-v6 { any; };
};
```

- Configure ***/etc/bind/named.conf.local***. This is the main configuration for DNS server. It is linked to the Forward Zone (*/etc/bind/open-ims.dnszone*) and Reverse Zone (*/etc/bind/open-ims-rev.dnszone*) file. In the Forward Zone, we define how to translate from the domain name open-ims.test to the IP address 127.0.0.1. In the Reverse Zone, we define the reverse translation from IP to domain name.

```
user@ubuntu# sudo gedit /etc/bind/named.conf.local

```

```
include "/etc/bind/rndc.key";
include "/etc/bind/zones.rfc1918";

controls {
  inet 127.0.0.1 port 953
  allow{127.0.0.1;}keys{"rndc-key";};
};

zone "open-ims.test" {
  type master;
  file "/etc/bind/open-ims.dnszone";
};

zone "0.0.127.in-addr.arpa" {
  type master;
  file "/etc/bind/open-ims-rev.dnszone";
};
```

- Create ***/etc/bind/open-ims.dnszone*** file as mentioned in the previous step. The original file is provided in the source code of IMS which is ***/opt/OpenIMSCore/ser_ims/cfg/open-ims.dnszone***. We make few modifications to adapt for our IMS test-bed system.

```
user@ubuntu# sudo gedit /etc/bind/open-ims.dnszone

$TTL 1W

@ 1D IN SOA localhost.open-ims.test. root.localhost.open-ims.test. (
            2006101001      ; serial
            3H              ; refresh
            15M             ; retry
            1W              ; expiry
            1D )            ; minimum

open-ims.test.  1D IN NS        localhost.open-ims.test.
open-ims.test.  1D IN A         127.0.0.1
localhost       1D IN A         127.0.0.1

pcscf              1D IN A         127.0.0.1
_sip.pcscf         1D SRV 0 0 4060 pcscf
_sip._udp.pcscf    1D SRV 0 0 4060 pcscf
_sip._tcp.pcscf    1D SRV 0 0 4060 pcscf

icscf              1D IN A         127.0.0.1
```

```
_sip                 1D SRV 0 0 5060 icscf
_sip._udp            1D SRV 0 0 5060 icscf
_sip._tcp            1D SRV 0 0 5060 icscf

open-ims.test. 1D IN A          127.0.0.1
open-ims.test. 1D IN NAPTR 10 50 "s" "SIP+D2U"  ""  _sip._udp
open-ims.test. 1D IN NAPTR 20 50 "s" "SIP+D2T"  ""  _sip._tcp

scscf                1D IN A          127.0.0.1
_sip.scscf           1D SRV 0 0 6060 scscf
_sip._udp.scscf      1D SRV 0 0 6060 scscf
_sip._tcp.scscf      1D SRV 0 0 6060 scscf

trcf                 1D IN A          127.0.0.1
_sip.trcf            1D SRV 0 0 3060 trcf
_sip._udp.trcf       1D SRV 0 0 3060 trcf
_sip._tcp.trcf       1D SRV 0 0 3060 trcf

bgcf                 1D IN A          127.0.0.1
_sip.bgcf            1D SRV 0 0 7060 bgcf
_sip._udp.bgcf       1D SRV 0 0 7060 bgcf
_sip._tcp.bgcf       1D SRV 0 0 7060 bgcf

mgcf                 1D IN A          127.0.0.1
_sip.mgcf            1D SRV 0 0 8060 mgcf
_sip._udp.mgcf       1D SRV 0 0 8060 mgcf
_sip._tcp.mgcf       1D SRV 0 0 8060 mgcf

hss                  1D IN A          127.0.0.1
ue                   1D IN A          127.0.0.1
presence             1D IN A          127.0.0.1
pcrf                 1D IN A          127.0.0.1
clf                  1D IN A          127.0.0.1
```

- Create ***/etc/bind/open-ims-rev.dnszone***

```
user@ubuntu# sudo gedit /etc/bind/open-ims-rev.dnszone

$TTL 1W

@ IN SOA localhost.open-ims.test. root.localhost.open-ims.test. (
            2006101001      ; serial
            3H              ; refresh
            15M             ; retry
            1W              ; expiry
            1D )            ; minimum

  IN  NS        localhost.

1  IN  PTR pcscf.open-ims.test.
1  IN  PTR icscf.open-ims.test.
1  IN  PTR scscf.open-ims.test.
1  IN  PTR trcf.open-ims.test.
1  IN  PTR bgcf.open-ims.test.
1  IN  PTR mgcf.open-ims.test.
1  IN  PTR hss.open-ims.test.
1  IN  PTR ue.open-ims.test.
1  IN  PTR presence.open-ims.test.
1  IN  PTR pcrf.open-ims.test.
1  IN  PTR clf.open-ims.test.
```

Note that ***1*** is the last octet of the IP address ***127.0.0.1***.

Start DNS Server

After configuring the DNS server, we need to restart the BIND9 service so that the configuration can take effect.

```
user@ubuntu# sudo service bind9 restart
```

Usually when we cannot start BIND9 service, we have mistyped the configuration in the three files ***named.conf***, ***named.conf.options*** and ***named.conf.local***. The structure of those files must follow certain formats as mention in [26]. To view more specific error details we can examine the system log. In most cases, the error will indicate exactly what components go wrong.

```
user@ubuntu# tail -f /var/log/syslog
```

Having the DNS service up and running we can check if the resolution between domain name and IP address is working as expected. The following results are extracted from a correctly working DNS resolution.

- Resolving domain name:

```
user@ubuntu# nslookup open-ims.test

Server:        127.0.0.1
Address:       127.0.0.1#53
```

```

Name:          open-ims.test
Address:       127.0.0.1
```

- Resolving IP address:

```
user@ubuntu# nslookup 127.0.0.1

Server:          127.0.0.1
Address:         127.0.0.1#53

1.0.0.127.in-addr.arpa  name = clf.open-ims.test.
1.0.0.127.in-addr.arpa  name = hss.open-ims.test.
1.0.0.127.in-addr.arpa  name = bgcf.open-ims.test.
1.0.0.127.in-addr.arpa  name = mgcf.open-ims.test.
1.0.0.127.in-addr.arpa  name = pcrf.open-ims.test.
1.0.0.127.in-addr.arpa  name = trcf.open-ims.test.
1.0.0.127.in-addr.arpa  name = icscf.open-ims.test.
1.0.0.127.in-addr.arpa  name = pcscf.open-ims.test.
1.0.0.127.in-addr.arpa  name = scscf.open-ims.test.
1.0.0.127.in-addr.arpa  name = presence.open-ims.test.
1.0.0.127.in-addr.arpa  name = ue.open-ims.test.
```

- Testing CSCFs and HSS connectivity by PING. They should be reachable:

```
user@ubuntu# ping pcscf.open-ims.test
user@ubuntu# ping icscf.open-ims.test
user@ubuntu# ping scscf.open-ims.test
user@ubuntu# ping hss.open-ims.test
```

We rarely have problems with forward resolving because our forward zone file is based on the sample IMS zone file. We sometimes have problems with reverse resolving. When the reverse resolution errors are SERFAIL and NXDOMAIN, our reverse zone file *open-ims-rev.dnszone* is not correctly defined. We should check the BIND9 manual [26].

C.4 Configure DNS Core Components

Up to this point we already have all the prerequisites for IMS components. Before proceeding to the following sections, we must make sure that MySQL and BIND9 service are running.

```
user@ubuntu# sudo service bind9 status
 * bind9 is running

```

```
user@ubuntu# sudo service mysql status
mysql start/running, process 1579
```

Change IP Address and Domain for IMS configuration files

- Go to directory ***opt/OpenIMSCore/ser_ims/cfg*** and run script ***configurator.sh***

  ```
  user@ubuntu# cd /opt/OpenIMSCore/ser_ims/cfg
  user@ubuntu# sh ./configurator.sh

  Domain Name: open-ims.test
  IP Address: 127.0.0.1
  ```

  These replace all the default IP Address and Domain Name by the specified values for the following files: ***icscf.cfg, icscf_pg.sql, icscf.sql, icscf.thig.cfg, icscf.xml, pcscf.cfg, pcscf.xml, persist_my.sql, persist_pg.sql, scscf.cfg, scscf.xml***.

- Go to directory ***/opt/OpenIMSCore/FHoSS/config/*** and change IP address and Domain Name in file ***DiameterPeerHSS.xml*** to our values. This file contains the configuration for FHoSS to connect to other peers.

  ```
  user@ubuntu# cd /opt/OpenIMSCore/FHoSS/config
  user@ubuntu# sudo gedit DiameterPeerHSS.xml

  <?xml version="1.0" encoding="UTF-8"?>
  <!-- HSS Server config -->
  <DiameterPeer
    FQDN="hss.open-ims.test"
    Realm="open-ims.test"
    Vendor_Id="10415"
    Product_Name="JavaDiameterPeer"
    AcceptUnknownPeers="1"
    DropUnknownOnDisconnect="1"
    Tc="30"
    Workers="4"
    QueueLength="32"
  >
    <Peer FQDN="icscf.open-ims.test" Realm="open-ims.test" port="3869"
      />
    <Peer FQDN="scscf.open-ims.test" Realm="open-ims.test" port="3870"
      />

    <Acceptor port="3868" bind="127.0.0.1" />

  ```

```
  <Auth id="16777216" vendor="10415"/><!-- 3GPP Cx -->
  <Auth id="16777216" vendor="4491"/><!-- CableLabs Cx -->
  <Auth id="16777216" vendor="13019"/><!-- ETSI/TISPAN Cx -->
  <Auth id="16777216" vendor="0"/><!-- ETSI/TISPAN Cx -->
  <Auth id="16777217" vendor="10415"/><!-- 3GPP Sh -->
  <Auth id="16777221" vendor="10415"/>

</DiameterPeer>
```

Compile CSCFs source code

There are five main modules in the source code. Three are the CSCF functionality modules and two are the interface modules for CSCFs to commute with each other and FHoSS.

- **P-CSCF Module** (Figure 45) provides many functions such as signalling firewall, service-route verification, IPsec association and Network Address Translation.
- **I-CSCF Module** (Figure 46) has full Cx interface support, topology hiding and network domain security capability.
- **S-CSCF Module** (Figure 47) has many features such as authentication (though AKAv1-MD5, AKAv2-MD5 and MD5), service-router and path header support.
- **CDiamterPeer (CDP) Module** is the interface for CSCFs to communicate with each other using Diameter protocol.
- **IMS Service Control (ISC) Module** provides the interface to communicate with Application Server (Asterisk and auSystems).

The source codes of CSCFs can be compiled with the following commands:

```
user@ubuntu# cd /opt/OpenIMSCore/ser_ims
user@ubuntu# make install-libs all
```

Compile FHoSS source code and Create User Database

FHoSS is written completely in Java. It communicates with I-CSCF and S-CSCF though Cx and Dx interface of CDP module. The Diameter commands are implemented upon FOKUS own Java based Diameter stack (Figure 48). The subscription information is stored in MySQL database and can be managed though web-pages. The following steps are used to compile FHoSS.

- Find and set JAVA_HOME location.

```
user@ubuntu# readlink -f /usr/bin/java | sed "s:jre/bin/java::"
/usr/lib/jvm/java-7-openjdk-i386/
user@ubuntu# export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386/
```

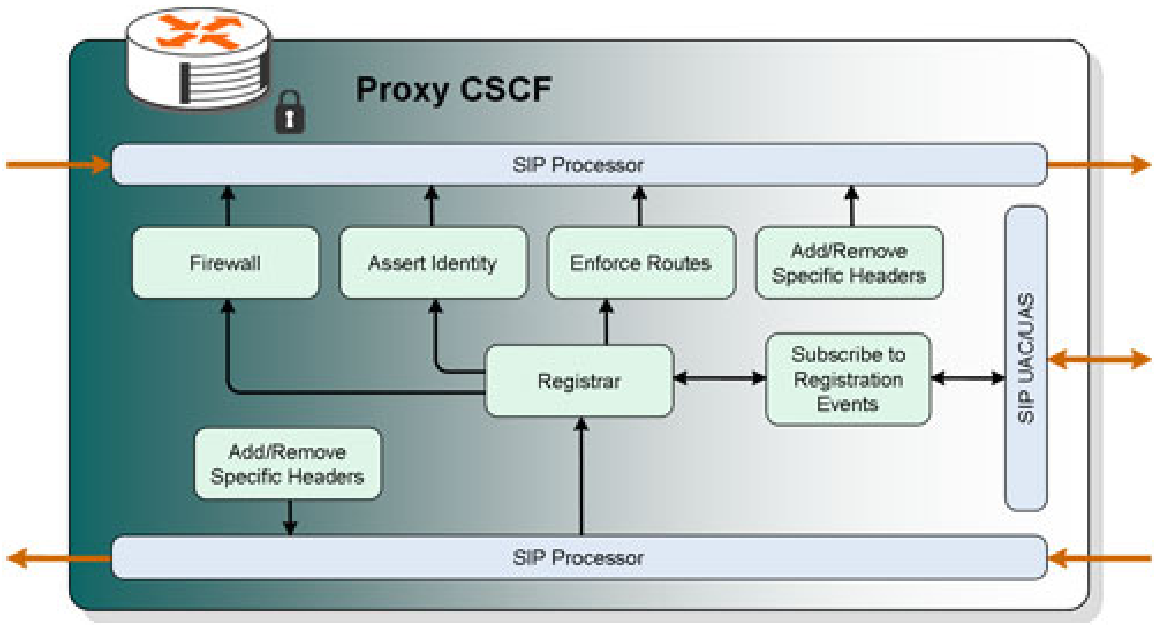


Figure 45. P-CSCF Module of Open Source IMS System.[4]

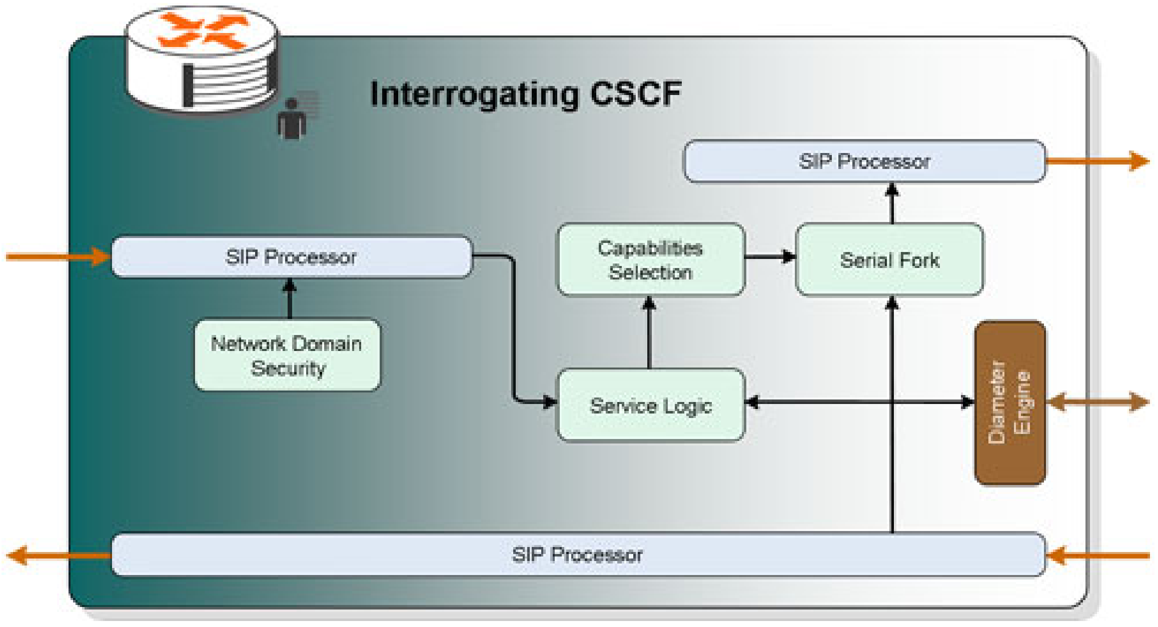


Figure 46. I-CSCF Module of Open Source IMS System.[4]

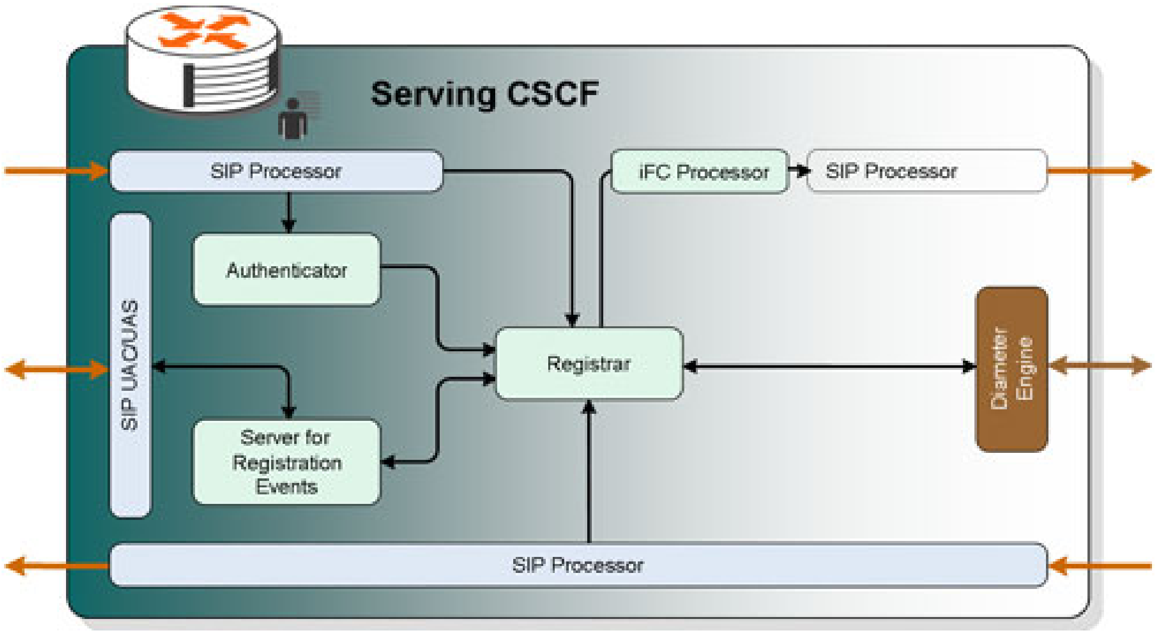


Figure 47. S-CSCF Module of Open Source IMS System.[4]

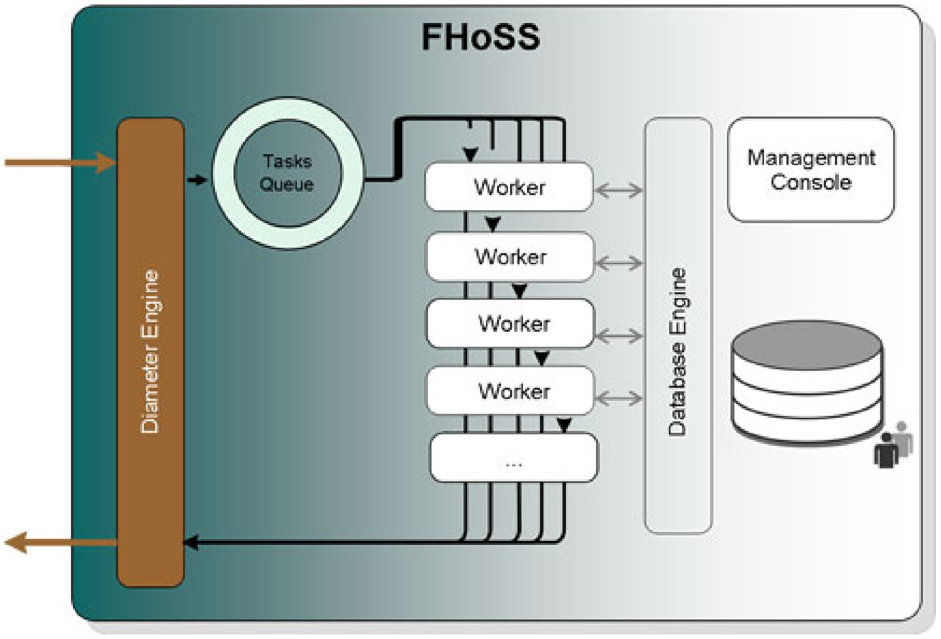


Figure 48. FHoSS Module of Open Source IMS System.[4]

- Compile FHoSS source code:

```
user@ubuntu# cd /opt/OpenIMSCore/FHoSS
user@ubuntu# ant compile deploy
```

- Create Databases for FHoSS and I-CSCF:

```
user@ubuntu# cd /opt/OpenIMSCore
user@ubuntu# mysql -u root -p < ser_ims/cfg/icscf.sql
user@ubuntu# mysql -u root -p < FHoSS/scripts/hss_db.sql
user@ubuntu# mysql -u root -p < FHoSS/scripts/userdata.sql
```

- Check if the database named ***hss_db*** and ***icscf*** are created:

```
user@ubuntu# mysql -u root -p
Enter password:

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| hss_db             |
| icscf              |
| mysql              |
| performance_schema |
+--------------------+
5 rows in set (0.01 sec)
```

The structure of ***hss_db*** and ***icscf*** are shown in Appendix E.

C.5 Start IMS Components

Up to this point, all CSCFs were successfully compiled. FHoSS were compiled and databases are created with two sample users named Alice and Bob. The following steps will help to start each component separately.

- Copy the configuration files **.cfg**, **.xml** and script files **.sh** to ***/opt/OpenIMSCore/***

```
user@ubuntu# cp ser_ims/cfg/*.cfg /opt/OpenIMSCore/
user@ubuntu# cp ser_ims/cfg/*.xml /opt/OpenIMSCore/
user@ubuntu# cp ser_ims/cfg/*.sh /opt/OpenIMSCore/
```

- Start CSCFs in 3 separate terminals
  - Terminal 1: Start P-CSCF

```
user@ubuntu# cd /opt/OpenIMSCore/
user@ubuntu#./pcscf.sh
```

- Terminal 2: Start I-CSCF

```
user@ubuntu# cd /opt/OpenIMSCore/
user@ubuntu#./icscf.sh
```

- Terminal 3: Start S-CSCF

```
user@ubuntu# cd /opt/OpenIMSCore/
user@ubuntu#./scscf.sh
```

- Start HSS in a separate terminal - The original command on [2] instructs to run **FHoSS/deploy/startup.sh**. But if we run this command when we are at ***/opt/OpenIMSCore/*** the script will fail to start.

```
user@ubuntu# cd /opt/OpenIMSCore/
user@ubuntu# sudo FHoSS/deploy/startup.sh

Classpath is lib/*.jar::log4j.properties:..
.............
Could not find the main class: de.fhg.fokus.hss.main.HSSContainer.
    Program will exit.
```

The script tries to build the ***CLASSPATH*** which is used by JAVA to run the program but it fails to locate the directory. The script looks into the current location for ***Lib*** directory. Under this location, it looks for all ***.jar*** files and then reads their names to append to the ***CLASSPATH***. Because the current locations is ***/opt/OpenIMSCore/***, there are no ***Lib*** directory and ***.jar*** files. It fails to build the ***CLASSPATH***, which leads to ***de.fhg.fokus.hss.main.HSSContainer*** class could not be found. Thus we have to run the script in a different location.

```
user@ubuntu# cd /opt/OpenIMSCore/FHoSS/deploy/
user@ubuntu# ./startup.sh

Classpath is lib/xml-apis.jar:........:log4j.properties:..
.......
Type "exit" to stop FHoSS!
```

- After all the components manage to start up, we can connect to the FHoSS to perform the provisioning through address *http://localhost:8080/* (Figure 49). The credential are:

Username:	hssAdmin
Password:	hss

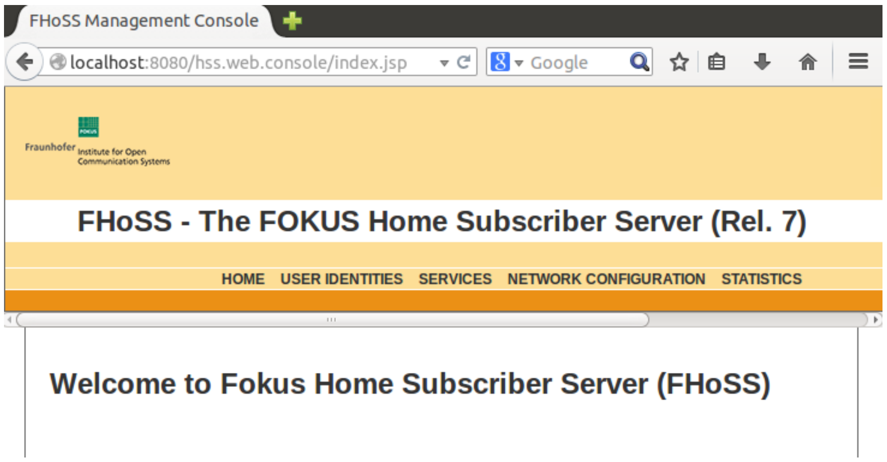

FHoSS Management Console
localhost:8080/hss.web.console/index.jsp
Google
Fraunhofer Institute for Open Communication Systems
FHoSS - The FOKUS Home Subscriber Server (Rel. 7)
HOME
USER IDENTITIES
SERVICES
NETWORK CONFIGURATION
STATISTICS
Welcome to Fokus Home Subscriber Server (FHoSS)

Figure 49. FHoSS Management Console.

After connecting to the FHoSS Management Console, we can see three main functionality tabs. The **User Identities** tab is for IMS subscription, private and public identity configuration. The **Services** tab is for Service Profiles, Application Servers, Initial Filter Criteria. The **Network Configuration** tab is for Visited Networks, Charging, Capability Sets and Preferred S-CSCF Sets.

To start IMS components easier, we create two scripts named **startIMS.sh** and **startIMS_log.sh**. They can start each IMS component in separate terminal tabs with logging capability. The Appendix A explains their details and usages.

D IMS Bench SIPp Implementation Steps

Install Prerequisites

- Adjust the kernel Timer Frequency from 250Hz to 1000Hz. The procedure is shown in Appendix G.
- Install **GNU Scientific Library**. This is required for random number generation for the statistical distribution.
  - Get the source code and compile

    ```
    user@ubuntu# cd /opt
    user@ubuntu# wget ftp://ftp.gnu.org/gnu/gsl/gsl-1.16.tar.gz
    user@ubuntu# tar -xzvf gsl-1.16.tar.gz
    user@ubuntu# cd gsl-1.16
    user@ubuntu# ./configure
    user@ubuntu# make
    user@ubuntu# make install
    ```

  - Add the path to the library to the **/etc/ld.so.conf** file

    ```
    user@ubuntu# echo /usr/local/lib/ >>/etc/ld.so.conf
    user@ubuntu# ldconfig
    ```

- Install **menu-driven benchmark configuration tool**. This is required for report generation.
  - Install **Perl XML::Simple** module
    - Install **expat** package

      ```
      user@ubuntu# apt-get install expat
      user@ubuntu# apt-get install libexpat1-dev
      ```

    - Install **XML::Parser, XML::SAX::Expat** and **XML::Simple** module.

      ```
      user@ubuntu# perl -MCPAN -e shell
      {reply with default answers... just select the local ftp
          server}
      cpan> install XML::Parser
      cpan> install XML::SAX::Expat
      cpan> install XML::Simple
      cpan> quit
      ```

  - Install **Gnuplot 4.2**

```
user@ubuntu# sudo apt-get install libgd2-xpm-dev
user@ubuntu# cd /opt
user@ubuntu# wget ftp://ftp.dante.de/pub/tex/graphics/gnuplot
    /4.6.5/gnuplot-4.6.5.tar.gz
user@ubuntu# tar -xzvf gnuplot-4.6.5.tar.gz
user@ubuntu# cd gnuplot-4.6.5
user@ubuntu# ./configure --without-x
user@ubuntu# make
user@ubuntu# make install
```

- Adjust the system limits to accommodate large number of sockets. These steps must be done as *root* user.
  - In **/etc/security/limits.conf** add:

    ```
    root@ubuntu# sudo gedit /etc/security/limits.conf
      * soft nofile 102400
      * hard nofile 409600
    ```

  - In **/etc/pam.d/logi**n add:

    ```
    root@ubuntu# sudo gedit /etc/pam.d/login
      session required /lib/security/pam_limits.so
    ```

  - In **/etc/sysctl.con**f add:

    ```
    root@ubuntu# sudo gedit /etc/sysctl.conf
      fs.file-max = 102400
      fs.inode-max = 409600
    ```

  - In **/root/.bashrc** add:

    ```
    root@ubuntu# sudo gedit /root/.bashrc
      ulimit -n 409600
    ```

  - Reboot the workstation for the modifications to take effect.

Install IMS Bench

- Download the source code.

```
user@ubuntu# cd /opt
user@ubuntu# svn co https://svn.code.sf.net/p/sipp/code/sipp/branches
    /ims_bench ims_bench
```

- Make sure to have both **gcc** and **gcc++** compiler installed.

```
user@ubuntu# apt-get install gcc
user@ubuntu# apt-get install g++
```

- Edit file **/opt/ims_bench/rmt/RmtDefs.hpp**

```
user@ubuntu# sudo gedit rmt/RmtDefs.hpp# apt-get install g++
include <cstddef>
```

- Install required packages to compile the program.

```
user@ubuntu# apt-get install libssl-dev
user@ubuntu# apt-get install libncurses5-dev
user@ubuntu# apt-get install libgsl0-dev
```

- Compile the program.

```
user@ubuntu# make rmtl
user@ubuntu# make ossl
user@ubuntu# make mgr
```

E Database Structure

The database structure of FHoSS and I-CSCF is shown below.

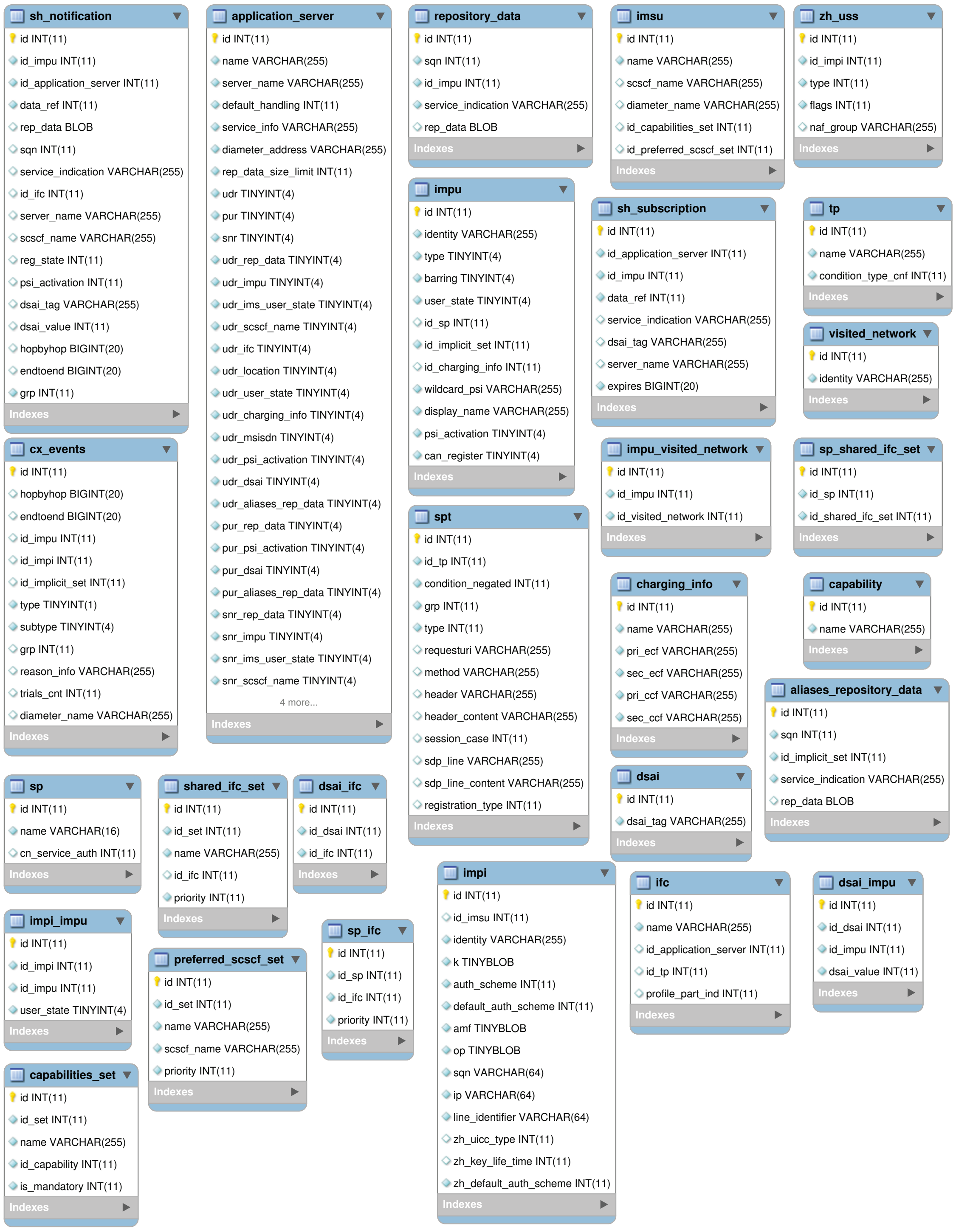


Figure 50. FHoSS Database Structure.

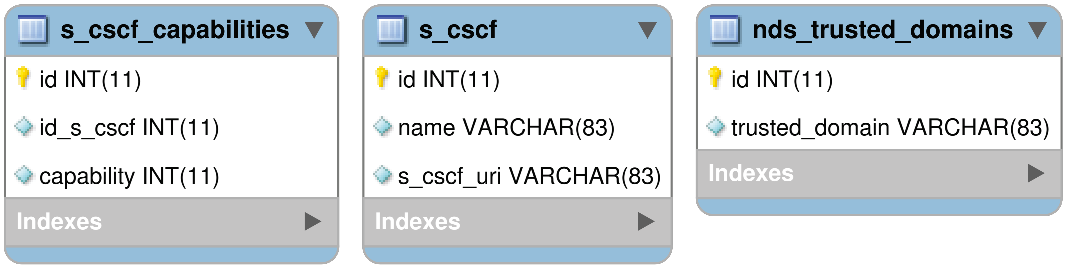


Figure 51. I-CSCF Database Structure.

F myMONSTER Installation

- Install the dependencies

```
user@ubuntu# sudo apt-get gstreamer0.10-plugins-bad
user@ubuntu# sudo apt-get libgstreamer0.10-dev
```

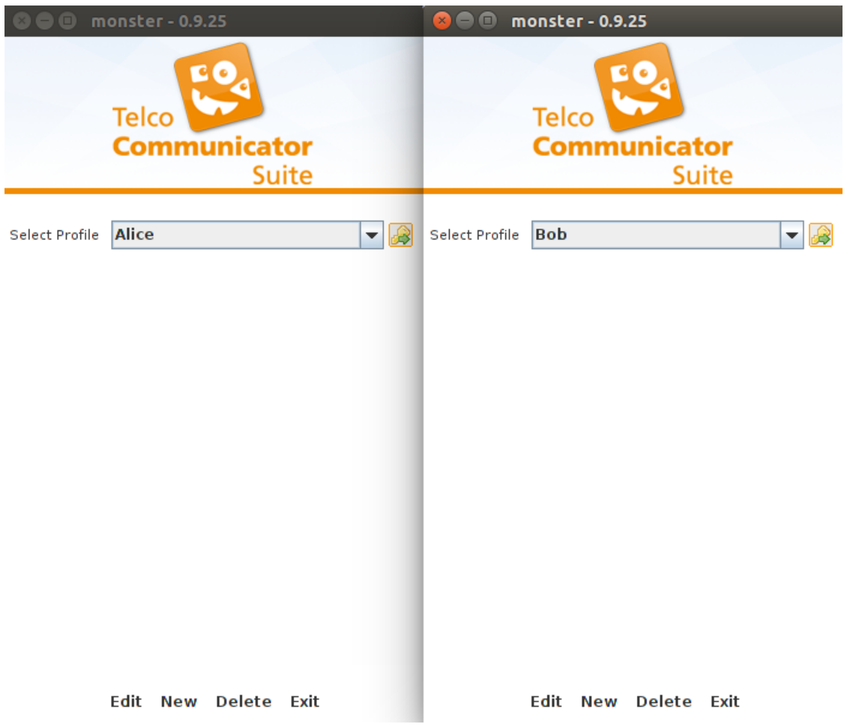
- Download myMONSTER

```
user@ubuntu# wget http://www.monster-the-client.org/downloads/
    download_TCS/_monster_downloads/myMONSTER-TCS_Linux32_v0_9_25_tar.
    gz
user@ubuntu# tar -xzvf myMONSTER-TCS_Linux32_v0_9_25_tar.gz
```

- Launch myMONSTER

```
user@ubuntu# cd ./monster-0.9.25
user@ubuntu# sudo ./monster
```

We launch another instance with the above process. We need two clients to test the basic communication.



Select Profile Alice

Select Profile Bob

Figure 52. myMONSTER Login Screen.

G Adjust Ubuntu Kernel Timer Frequency

- Check the current Timer Frequency

```
user@ubuntu# cat /boot/config-`uname -r` | grep HZ
.........
# CONFIG_HZ_100 is not set
CONFIG_HZ_250=y
# CONFIG_HZ_300 is not set
# CONFIG_HZ_1000 is not set
CONFIG_HZ=250
.........
```

- Install required packages for kernel compilation.

```
user@ubuntu# su
user@ubuntu# apt-get update
user@ubuntu# apt-get install kernel-package libncurses5-dev fakeroot
    wget bzip2
```

- Download the kernel sources.

```
user@ubuntu# cd /usr/src
user@ubuntu# wget https://www.kernel.org/pub/linux/kernel/v3.x/linux
    -3.13.11.tar.gz
```

- Unpack the kernel and create a link to the kernel source directory.

```
user@ubuntu# tar -xzvf linux-3.13.11.tar.gz
user@ubuntu# ln -s linux-3.13.11 linux
```

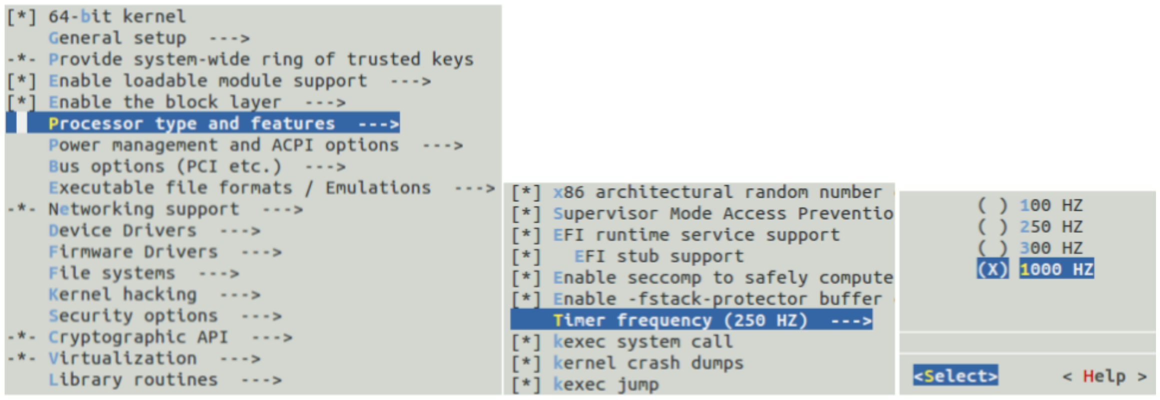
- Adjust the kernel timer frequency.

```
user@ubuntu# cp /boot/config-`uname -r` ./.config
user@ubuntu# make menuconfig
```

  This brings up **Kernel Configuration Menu** (Figure 53). Navigate to **Process type and features**, then **Timer frequency**. Check the frequency **1000Hz**. Finally Save and Exit the Menu.

- Build the kernel.

```
user@ubuntu# make-kpkg clean
user@ubuntu# fakeroot make-kpkg --initrd --append-to-version=-custom
    kernel_image kernel_headers
```



```
[*] 64-bit kernel
    General setup  --->
-*- Provide system-wide ring of trusted keys
[*] Enable loadable module support  --->
[*] Enable the block layer  --->
    Processor type and features  --->
    Power management and ACPI options  --->
    Bus options (PCI etc.)  --->
    Executable file formats / Emulations  --->
-*- Networking support  --->
    Device Drivers  --->
    Firmware Drivers  --->
    File systems  --->
    Kernel hacking  --->
    Security options  --->
-*- Cryptographic API  --->
-*- Virtualization  --->
    Library routines  --->
```

```
[*] x86 architectural random number
[*] Supervisor Mode Access Preventio
[*] EFI runtime service support
[*]   EFI stub support
[*] Enable seccomp to safely compute
[*] Enable -fstack-protector buffer
    Timer frequency (250 HZ)  --->
[*] kexec system call
[*] kernel crash dumps
[*] kexec jump
```

```
( ) 100 HZ
( ) 250 HZ
( ) 300 HZ
(X) 1000 HZ
```

Figure 53. Kernel Timer Frequency Adjustment.

After this there are two .deb files

```
# cd /usr/src
# ls -l
linux-headers-3.13.11.deb
linux-image-3.13.11.deb
```

- Install the kernel.

```
# sudo dpkg -i linux-headers-3.13.11.deb linux-image-3.13.11.deb
```

Restart the workstation to complete the kernel installation.

H Script to Generate User List

The original **user_gen.pl** script is located in **/opt/ims_bench**. It is modified to use with our IMS test-bed. This script does not create users for FHoSS. It only generates a list of usernames with few subscription information for IMS Bench SIPp.

```
#!/usr/bin/perl

if (@ARGV < 2) {
  die("Required arguments: <start_counter> <nbUsers>");
}

$start = $ARGV[0];
$nbUsers = $ARGV[1];
$domain = "open-ims.test";
$realm = "open-ims.test";
$user_prefix = "subs";
$usim_prefix = "subs";
$initial_pool = 0;

for($i=$start; $i<($start+$nbUsers); ++$i) {
    printf("%d;%s%06d;%s;%s%06d;%s;%s%06d;data%d_1\n", $initial_pool,
    $user_prefix, $i, $domain, $usim_prefix, $i, $realm, $user_prefix, $i, $i);
}
```

How to use the script?

```
user@ubuntu# ./user_gen.pl 1 1000 > ims_users_1.inf
```

A file named ims_users_1.inf with user info are created with the content

```
0;subs000001;open-ims.test;subs000001;open-ims.test;subs000001;data1_1
0;subs000002;open-ims.test;subs000002;open-ims.test;subs000002;data2_1
0;subs000003;open-ims.test;subs000003;open-ims.test;subs000003;data3_1
0;subs000004;open-ims.test;subs000004;open-ims.test;subs000004;data4_1
0;subs000005;open-ims.test;subs000005;open-ims.test;subs000005;data5_1
......
```

It has 7 data columns separated by semicolon.

- Column 1 → User Pool
- Column 2 → User Public ID

- Column 3 → Domain
- Column 4 → User Private ID
- Column 5 → Realm
- Column 6 → Password
- Column 7 → Extra Data (not use in our case)

I XML Configuration For SIPp Manager

The file manager.xml is the configuration specifying what scenarios the test case uses.

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>

  <!-- Test System Parameters -->
  <param name="number_test_systems" value="0"/>
  <param name="prep_offset" value="2000"/>
  <param name="rand_seed" value="0"/>
  <param name="report" value="1"/>
  <param name="log" value="1"/>
  <param name="transient_time" value="1"/>
  <param name="max_time_offset" value="250"/>

  <!-- Scenario Parameters -->
  <scen_param name="HoldTime" value="30000"/>
  <scen_param name="RingTime" value="5000"/>
  <scen_param name="RegistrationExpire" value="1000000"/>

  <!-- List all the scenario that needs to be loaded -->
  <param name="scenario_path" value="scen"/>
  <scenario name="ims_uac"   max_ihs="0.1"/>
  <scenario name="ims_uas"/>
  <scenario name="ims_reg"   max_ihs="0.1"/>
  <scenario name="ims_dereg" max_ihs="0.1"/>
  <scenario name="ims_rereg" max_ihs="0.1"/>
  <scenario name="ims_msgc"  max_ihs="0.1"/>
  <scenario name="ims_msgs"/>

  <!-- Pre-Registration phase - Default is "single step" -->
  <run cps="100" max_calls="1600" distribution="constant" sync_mode="off"
    use_scen_max_ihs="no" max_global_ihs="1" stats="1000">
  <scenario name="ims_reg" ratio="100"/>
</run>

<!-- Sleep -->
  <run cps="0" duration="3"/>

<!-- Stir phase to warm up the SUT -->
<run cps="40" duration="75" step_increase="20" num_steps="3" distribution="
    poisson" use_scen_max_ihs="no" max_global_ihs="1" stats="2000" report="no">
  <scenario name="ims_reg" ratio="2.5"/>
  <scenario name="ims_uac" ratio="50"/>
  <scenario name="ims_dereg" ratio="2.5"/>
  <scenario name="ims_msgc" ratio="30"/>
  <scenario name="ims_rereg" ratio="15"/>
</run>

<!-- Actual benchmark phase -->
<run cps="100" duration="60" step_increase="10" num_steps="0" distribution="
    poisson" stats="2000">
</run>

<!-- Done... Sleep for some more time -->
<run cps="0" duration="3"/>

</configuration>
```

What do the parameter mean?

Test System Parameters

number_test_systems	Manager waits until this many connecting instances to start generating load. 0 means any.
prep_offset	Time (in seconds) allocated for the preparation portion.
rand_seed	Seed value that is used for number generator. 0 means random.
report	Use to generate report. 1 = generate; 0 = do not generate.
log	Use to generate Manager logging (manager.log file). 1 = enable; 0 = disable.
transient_time	Time (in seconds) that HIS is ignored.
max_time_offset	Maximum offset (in microseconds) allowed between each TS and the Manager.

Scenario Parameters

We define the value of global generic parameters so that the scenarios can refer to. For example, in a pause command:

```
<pause poisson="true" mean="%RingTime"/>
```

Scenario List

scenario_path	Location of scenario files .xml.
max_ihs	Maximum percentage of IHS allowed for this scenario (in client side).

The scenario names must match with files names (.xml) in the scenario file folder

Pre-Registration Phase

This phase registers some users to use for the later phases.

cps Call rate. Use 0 for Pause phase.

max_calls Number of call to generate for this step.

distribution Distribution type: "constant" or "poisson".

use_scen_max_ihs Specify how to use the IHS values.

- "yes" = use the scenarios max_ihs value defined in CONFIGURATION section.
- "no" = use the max_global_ihs that is defined following.

max_global_ihs Maximum allowed percentage of IHS for the run.

stats Interval (in milliseconds)that the Manager queries SIPp instances for information to display.

ratio A run includes many scenarios. Each scenario has a ratio of occur.

- The total of the ratio of all scenarios must be 100.
- If one scenario is included. It stays for the following runs. Only its ratio in the following runs can be changes.

Sleep

The test case will be paused for some times.

Stir Phase

This phase do a warmup before running the real test.

duration	Duration a a single step of a run.
step_increase	The increase load (call rate) when moving to a next step of a run.
num_steps	How many steps a run includes.
report	Specify what to include in the report.

- "yes" = include this step to the report.
- "no" = not include this step to the report.

Actual Benchmark Phase

Even though the scenarios are not defined, this phase will run with the scenarios (with ratio) specified in the Stir Phase.

J Scripts to Add Users to FHoSS

Script to Add One User

The original script to add a user is located at **/opt/OpenIMSCore/ser_ims/cfg/add-imscore-user_newdb.sh**. We modify the script with the following changes such that we can reuse it to add multiple users.

```
.........

# Add MySQL password so that the process won't ask for every DB transaction
DBPASS=123456

.........

# Add user subscription information
CREATE_SCRIPT_TEMPLATE="


insert into hss_db.imsu(
  name,
  scscf_name,
  diameter_name,
  id_capabilities_set,
  id_preferred_scscf_set)
values ('<USER>',
  '',
  '',
  1,
  1);

insert into hss_db.impi(
        identity,
        id_imsu,
        k,
        auth_scheme,
        default_auth_scheme,
        amf,
        op)
values( '$IMPI',
        (select id from hss_db.imsu where hss_db.imsu.name='<USER>'),
        '$PASSWORD',
        127,
        1,
        '\0\0',
        '\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0');

insert into hss_db.impu(identity,id_sp,id_charging_info) values ('$IMPU', (
    select id from hss_db.sp order by id limit 1), 1);
update hss_db.impu set id_implicit_set=id where hss_db.impu.identity='$IMPU';

insert into hss_db.impi_impu(id_impi,id_impu) values ((select id from hss_db.
    impi where hss_db.impi.identity='$IMPI'), (select id from hss_db.impu where
    hss_db.impu.identity='$IMPU'));

insert into hss_db.impu_visited_network(id_impu, id_visited_network) values((
    select id from hss_db.impu where hss_db.impu.identity='$IMPU'), (select id
    from hss_db.visited_network where hss_db.visited_network.identity='$REALM'))
    ;
"

CREATE_TELURI_IMPU_TEMPLATE="

insert into hss_db.impu(identity,id_sp,id_charging_info) values ('$TELURI', (
    select id from hss_db.sp order by id limit 1), 1);
select @id:=id from hss_db.impu where hss_db.impu.identity='$IMPU';
update hss_db.impu set id_implicit_set=@id where hss_db.impu.identity='$TELURI';

insert into hss_db.impi_impu(id_impi,id_impu) values ((select id from hss_db.
    impi where hss_db.impi.identity='$IMPI'), (select id from hss_db.impu where
    hss_db.impu.identity='$TELURI'));

insert into hss_db.impu_visited_network(id_impu, id_visited_network) values((
    select id from hss_db.impu where hss_db.impu.identity='$TELURI'), (select id
     from hss_db.visited_network where hss_db.visited_network.identity='$REALM')
    );
"

DELETE_SCRIPT_TEMPLATE="
delete from hss_db.impu_visited_network where id_impu = (select id from hss_db.
    impu where hss_db.impu.identity='$IMPU');
delete from hss_db.impi_impu where id_impi = (select id from hss_db.impi where
    hss_db.impi.identity='$IMPI');
delete from hss_db.impu where identity = '$IMPU';
delete from hss_db.imsu where name = '<USER>';

.........

```

```
# The DB transaction will not ask for password anymore.
if [ $OPTION_ADD -eq 1 ]; then
    echo Apply $CREATE_SCRIPT as user $DBUSER...
    mysql -u$DBUSER -p$DBPASS < $CREATE_SCRIPT > /dev/null
    EXIT_CODE=$?
    SCRIPT=$CREATE_SCRIPT
elif [ $OPTION_DELETE -eq 1 ]; then
    echo Apply $DELETE_SCRIPT as user $DBUSER...
    mysql -u$DBUSER -p$DBPASS < $DELETE_SCRIPT
    EXIT_CODE=$?
    SCRIPT=$DELETE_SCRIPT
fi

.........
```

How to use the script?

```
user@ubuntu# ./add-imscore-user_newdb.sh -u <username> [-a|-d]
```

where

-u Username.

-p Password. If not specified, it is set to be the same as Username.

-r Realm. If not specified, it is set to be open-ims.test

-a Add User.

-d Delete User.

-c Clean the .sql file which is used for adding/deleting user.

When the script runs, it creates two SQL scripts named **add-user-<Username>.sql** and **delete-user-<Username>.sql**. Then according to the specified option -a or -d, it will run the add or delete script only. If -c option is specified, the SQL scripts will be deleted after running.

Script to Add Multiples User

We created a script named **multiusers.sh** to add a batch of users into the database. It is based on the original script **/opt/OpenIMSCore/ser_ims/cfg/add-imscore-user_newdb.sh**. We have to put these scripts in the same directory.

```bash
#!/bin/bash

Usage()
{
    echo "ERROR: Invalid parameters"
    echo "add-multiuser.sh -s <start_index> -e <stop_index> [-a|-d]"
    echo "  -s <start_index>: User Index to start."
    echo "  -e <stop_index>: User Index to end."
    echo "  -a: Add users."
    echo "  -d: Delete users."
    exit 1
}

ADD=0
DELETE=0
USER_PREFIX="subs"

while getopts s:e:ad?:? option;
do
    case $option in
        s) START=$OPTARG;;
        e) END=$OPTARG;;
        a) ADD=1;;
        d) DELETE=1;;
    esac
done

# Check if there are START and END index.
[ -z "$START" ] || [ -z "$END" ] && Usage;

# Check if ONLY ADD or DELETE is enable.
[ $ADD -eq 1 ] && [ $DELETE -eq 1 ] && Usage;

if [ $ADD -eq 1 ]; then
  while [ $START -le $END ]
  do
    USER_NAME=$(printf %s%06d $USER_PREFIX $START);
    ./add-imscore-user_newdb.sh -u $USER_NAME -a -c
    (( START++ ))
  done;

elif [ $DELETE -eq 1 ]; then
  while [ $START -le $END ]
  do
    USER_NAME=$(printf %s%06d $USER_PREFIX $START);
    ./add-imscore-user_newdb.sh -u $USER_NAME -d -c
    (( START++ ))
  done;
fi
```

How to use the script?

```
user@ubuntu# ./multiuser.sh -s <start_index> -e <start_index> [-a|-d]
```

where

-s Start index of Username.

-e End index of Username.

-a Add User.

-d Delete User.

Example

- This will add 900 users (1000-100) from the name *subs000100* to *subs001000*.

```
user@ubuntu## ./multiuser.sh -s 100 -e 1000 -a
```

- This will delete 200 user (400-200) from the name *subs000200* to *subs000400*.

```
user@ubuntu## ./multiuser.sh -s 200 -e 400 -a
```

K Test-bed Specific Configurations and Scenarios

The following configuration (**manager_reg.xml**) shows how we configure the SIPp Manager for the registration procedure.

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>

<configuration>
  <!-- Test System Parameters -->
  <param name="number_test_systems" value="0"/>
  <param name="prep_offset" value="2000"/>
  <param name="rand_seed" value="0"/>
  <param name="report" value="1"/>
  <param name="log" value="1"/>
  <param name="transient_time" value="1"/>

  <!-- Scenario Parameters -->
  <scen_param name="HoldTime" value="30000"/>
  <scen_param name="RingTime" value="5000"/>
  <scen_param name="RegistrationExpire" value="1000000"/>

  <!-- Registration Scenario -->
  <param name="scenario_path" value="scen"/>
  <scenario name="test_reg" max_ihs="0.1"/>

</configuration>

<!-- Actual Running -->
<run cps="30" duration="60" step_increase="10" num_steps="0" distribution="
    poisson" use_scen_max_ihs="no" max_global_ihs="1" stats="2000">
  <scenario name="test_reg" ratio="100"/>
</run>
```

The registration scenario (**test_reg.xml**) used for our SIPp Instance are modified from the sample registration scenario provided by IMS Bench SIPp.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<scenario name="test_reg" on_unexpected="9" default_behavior="false">
  <info>
    <metric ref="PX_TRT-REG1" rtd="1" max="2000"/>
    <metric ref="PX_TRT-REG2" rtd="2" max="2000"/>
  </info>
<!-- *** Pick Users to Testing *** -->
  <nop>
    <action>
      <assign_user pool="0" scheme="rand_uni"/>
      <move_user pool="1"/>
    </action>
  </nop>

<!-- *** Start Timer *** -->
  <sync crlf="true">
    <action>
      <exec int_cmd="set_start_time"/>
    </action>
  </sync>

<!-- *** STEP 3 *** -->
  <send retrans="500" start_rtd="1">
    <![CDATA[
      REGISTER sip:[field1] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: "[field0]" <sip:[field0]@[field1]>;tag=[call_number]
      To: "[field0]" <sip:[field0]@[field1]>
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Contact: <sip:[field0]@[local_ip]:[local_port]>;expires=[%
    RegistrationExpire]
      Expires: [%RegistrationExpire]
      Content-Length: 0
      Authorization: Digest username="[field2]@[field3]", realm="[field3]"
      Supported: path
    ]]>
  </send>

  <recv response="401" auth="true" auth_assign_to="u2" rtd="1">
  </recv>

  <send retrans="500" start_rtd="2">
    <![CDATA[
      REGISTER sip:[field1] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: "[field0]" <sip:[field0]@[field1]>;tag=[call_number]
      To: "[field0]" <sip:[field0]@[field1]>
      Call-ID: [call_id]
      CSeq: 2 REGISTER
      Contact: <sip:[field0]@[local_ip]:[local_port]>;expires=[%
    RegistrationExpire]
      Expires: [%RegistrationExpire]
      Content-Length: 0
      [authentication username=[field2]@[field3] password=[field4]]
      Supported: path
    ]]>
  </send>

  <recv response="200" rtd="2" crlf="true" next="10">
    <action>
      <ereg regexp=".*" search_in="hdr" header="Service-Route:" check_it="true"
    assign_to="u1" />
       <move_user pool="2"/>
    </action>
  </recv>

<!-- *** If testting fails, jump here *** -->
  <label id="9"/>

  <nop>
    <action>
      <move_user pool="0"/>
    </action>
  </nop>

  <label id="10"/>

<!-- *** Scale to display timer results *** -->
<ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>
<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>
</scenario>
```

A few important concepts for IMS Bench SIPp are:

- It reserves users from the initial pool (by select users from the pool and move them to other pool).
- It sends a SIP message (**<send>...</send>**) with the timer on (**start_rtd="..."**)
- It receives a SIP message (**<recv>...</recv>**) with timer off (**rtd="..."**). The received SIP message can be either optional (**<recv ... optional="true">**) or must (no optional field). An optional message means that the SIPp instance does not care about this message.
- The trip time is calculated and store in the RTD variable.
- SIPp instances communicate with each other using non-SIP message: they send messages by (**<sendRmt>...</sendRmt>**) and receive messages by (**<recvRmt>... </recvRmt>**). These are used to exchange timing info, user reservations and especially important in the case of client and server scenario (one user calls another user). The parameter of **<sendRmt>** will trigger the correct scenario for the server side.

L How to Collect IMS Bench SIPp Results

Gathering the Results

After running the test cases, the SIPp manager generates a report file **report.xml** containing start time, scenario attempt rates, etc. Then we copy this file to directory **/opt/ims_bench/scripts**. Each SIPp Instance dumps the results locally where it is running (if SIPp instances run on many different machines, the results are located on many different physical machines). They are **.csv** files: *sipp_TS<ts_id>_scen.csv* and *sipp_TS<ts_id>_retrans.csv*. There are 2 ways to collect them:

- Manually collect all of CSV files and put them into directory **/opt/ims_bench/scripts** of the Manager. Then merge the results.

  ```
  user@ubuntu# cd /opt/ims_bench/scripts
  user@ubuntu# sudo ./getResults.pl -merge
  user@ubuntu# sudo ./getResults.pl -clean
  ```

  This generates sipp.csv and sipp_retrans.csv. We clean the partial result files after this operation.

- Automatically collect and merge the result.

  ```
  user@ubuntu# cd /opt/ims_bench/scripts
  user@ubuntu# sudo ./getResults.pl
  ```

  This reads the manager report file to learn the IP addresses of the test systems and the PID or TS ID of their SIPp Instances. Then it collects the corresponding files using SCP and merges them to make the final reports: sipp.csv and sipp_retrans.csv.

Generating Report

- Copy all the scenarios for our test case into **/opt/ims_bench/scripts**.

- Generate the reports.

  ```
  user@ubuntu# sudo ./doReport.pl
  ```

  This generates *report.mht*, *report.html* (including many images .PNG) and data file .DAT. The file *report.mht* (which includes everything) can be opened with Internet Explorer. The file *report.html* (with its images) can be open with any browsers.

M Matlab Code to Calculate Arrival Rate

The following Matlab code named **ims_eval.m** is used to calculate the arrival rate for our IMS test-bed.

```matlab
% External Arrival Rate
lambda0 = 30;

% Transition Probability from External
Q = zeros(1,15);
Q(1,1) = 1;

% Identity Martix
I = eye(15);

% Transtion Probability Matrix
theta = zeros(15);
for i = 1:10
    theta(i,i+1) = 1;
end
theta(11,12) = 0.5;
theta(11,14) = 0.5;
theta(12,13) = 1;
theta(14,15) = 1;

% Arrival rate matrix
lambda = mrdivide(lambda0*Q,I-theta);
% Node P-CSCF
lambda_P = lambda(1) + lambda(13) + lambda(15);
% Node I-CSCF
lambda_I = lambda(2) + lambda(6) + lambda(12) + lambda(14);
% Node S-CSCF
lambda_S = lambda(7) + lambda(11);
% Node FHoSS
lambda_F = lambda(3) + lambda(5) + lambda(8) + lambda(10);
% Node MySQL
lambda_MySQL = lambda(4) + lambda(9);
```

N Matlab Code to Calculate Saturation Points

The following Matlab code named **servtime_1_machine.m** is used to get the saturation point at four IMS node components.

```
clc;
clear;
close all;

%################### P-CSCF ###################%
lambda = [5 10 15 20 25 30];
lambda_P = [0.0066 0.0132 0.0199 0.0265 0.0331 0.0397];
mu_P = [0.0241 0.0286 0.0324 0.0365 0.0408 0.0447];

%fit linear polynomial
p1 = polyfit(lambda,lambda_P,1); display(p1);
p2 = polyfit(lambda,mu_P,1);

%calculate intersection
x_intersect = fzero(@(x) polyval(p1-p2,x),3);
y_intersect = polyval(p1,x_intersect);

subplot(2,2,1);
line(lambda,lambda_P);
hold on;
line(lambda,mu_P,'LineStyle',':');
legend('\lambda_P','\mu_P');
plot(x_intersect,y_intersect,'r*')
xlabel('\lambda');
ylabel('s^-^1');
title('P-CSCF');

%################### I-CSCF ###################%
lambda = [5 10 15 20 25 30];
lambda_I = [0.0033 0.0066 0.0099 0.0132 0.0165 0.0198];
mu_I = [0.0167 0.0196 0.0221 0.0243 0.0265 0.0287];

%fit linear polynomial
p1 = polyfit(lambda,lambda_I,1); display(p1);
p2 = polyfit(lambda,mu_I,1);

%calculate intersection
x_intersect = fzero(@(x) polyval(p1-p2,x),3);
```

```
y_intersect = polyval(p1,x_intersect);

subplot(2,2,2);
line(lambda,lambda_I);
hold on;
line(lambda,mu_I,'LineStyle',':');
legend('\lambda_I','\mu_I');
plot(x_intersect,y_intersect,'r*')
xlabel('\lambda');
ylabel('s^-^1');
title('I-CSCF');

%################### S-CSCF ####################%
lambda = [5 10 15 20 25 30];
lambda_S = [0.0115 0.0231 0.0346 0.0461 0.0577 0.0692];
mu_S = [0.0378 0.0437 0.0517 0.0584 0.0642 0.0689];

%fit linear polynomial
p1 = polyfit(lambda,lambda_S,1); display(p1);
p2 = polyfit(lambda,mu_S,1);

%calculate intersection
x_intersect = fzero(@(x) polyval(p1-p2,x),3);
y_intersect = polyval(p1,x_intersect);

subplot(2,2,3);
line(lambda,lambda_S);
hold on;
line(lambda,mu_S,'LineStyle',':');
legend('\lambda_S','\mu_S');
plot(x_intersect,y_intersect,'r*')
xlabel('\lambda');
ylabel('s^-^1');
title('S-CSCF');

%################### FHoSS ####################%
lambda = [5 10 15 20 25 30];
lambda_F = [0.0030 0.0059 0.0089 0.0119 0.0148 0.0178];
mu_F = [0.0146 0.0177 0.0200 0.0219 0.0236 0.0254];

%fit linear polynomial
p1 = polyfit(lambda,lambda_F,1); display(p1);
p2 = polyfit(lambda,mu_F,1);

```

```
%calculate intersection
x_intersect = fzero(@(x) polyval(p1-p2,x),3);
y_intersect = polyval(p1,x_intersect);

subplot(2,2,4);
line(lambda,lambda_F);
hold on;
line(lambda,mu_F,'LineStyle',':');
legend('\lambda_F','\mu_F');
plot(x_intersect,y_intersect,'r*')
xlabel('\lambda');
ylabel('s^-^1');
title('FHoSS');
```

O Matlab Code to Calculate CPU Utilization

The following Matlab code named **cpu.m** is used to get the CPU utilization for four IMS node components.

```matlab
clc;
clear;
close all;

% Component Utilization
lambda = [5 10 15 20 25 30 35 40];
U_P = [4.98 9.87 14.76 19.94 24.73 29.13 34.30 39.84];
U_S = [5.67 11.23 16.38 22.69 27.83 33.36 38.46 44.92];
U_I = [3.24 6.58 9.72 12.76 16.60 19.94 22.68 25.72];
U_F = [2.66 5.62 8.28 11.44 13.8 16.46 19.32 22.38];

% Fit linear polynomial
m_P = polyfit(lambda,U_P,1);
m_S = polyfit(lambda,U_S,1);
m_I = polyfit(lambda,U_I,1);
m_F = polyfit(lambda,U_F,1);

hold on;
line(lambda,U_P,'LineStyle','-','Color','r');
line(lambda,U_S,'LineStyle','--','Color','g');
line(lambda,U_I,'LineStyle',':','Color','b');
line(lambda,U_F,'LineStyle','-.','Color','m');
legend('P-CSCF','S-CSCF','I-CSCF','F-CSCF');
xlabel('\lambda');
ylabel('%CPU');
```

P Matlab Code to Calculate Confidence Interval

The following Matlab code named **confidence_interval.m** is used to calculate the confidence interval of the recorded serving time at the IMS nodes. The given code is applied for P-CSCF only but we can change the input data for the appropriate IMS nodes.

```matlab
clc;
clear all;

% The data
y = [39.198 44.691 43.986 46.326 43.207];
% y = [38.644 39.124 40.227 39.782 40.728];
% y = [36.124 36.312 36.870 36.295 36.742];
% y = [32.546 32.129 32.476 32.842 32.617];

% The average and Standard deviation
ybar = mean(y);
s = std(y);

% The Confidence Interval
ci = 0.98;
alpha = 1 - ci;

n = length (y);
T_multi = tinv(1-alpha/2, n-1);
ci_range = T_multi*s/sqrt(n);

sprintf('The confidence interval is between %1.3f and %1.3f', ybar - ci_range,
    ybar + ci_range)
```

References

[1] L. Nagy, J. Hosek, P. Vajsar, and V. Novotny, “Impact of signalling load on response times for signalling over IMS core,” in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, pp. 663–666, IEEE, 2013.

[2] “Open Source IMS Core System.” http://www.openimscore.org/.

[3] “IMS Bench SIPp.” http://sipp.sourceforge.net/ims_bench/.

[4] “Open IMS Playround.” http://www.fokus.fraunhofer.de.

[5] M. Handley and V. Jacobson, “RFC 2327,” *SDP: session description protocol*, vol. 10, 1998.

[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “RFC 3261: SIP: Session Initiation Protocol,” 2003.

[7] “SingTel and Ericsson unveil worlds first nationwide full featured voice over LTE.” http://www.ericsson.com/news/140521-singtel-and-ericsson-unveil-worlds-first-nationwide-full-featured-voice-over-lte.

[8] “3GPP Features and Study Items.” http://www.3gpp.org/DynaReport/FeatureListFrameSet.htm.

[9] *IP Multimedia Subsystem (IMS); Stage 2*, 12 2013. TS 23.228 version 11.10.0 Release 11.

[10] P. Faltstrom, “RFC 2916 - E. 164 number and DNS,” 2000.

[11] B. Aboba and M. Beadles, “RFC 2486: The Network Access Identifier,” *Network Working Group*, 1999.

[12] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko, “RFC 3588-Diameter Base Protocol,” *Network Working Group*, p. 48, 2003.

[13] L. Kleinrock, "Queueing systems. volume 1: Theory," 1975.

[14] "IMS/NGN Performance Benchmark, Technical Standard," *ETSI TS*, vol. 186, no. 008, 2012.

[15] G. Din, R. Petre, and I. Schieferdecker, "A Workload Model for Benchmarking IMS Core Networks," in *GLOBECOM*, pp. 2623–2627, 2007.

[16] R. Herpertz and J. M. E. Carlin, "A Performance Benchmark of a Multimedia Service Delivery Framework," in *Computer Science (ENC), 2009 Mexican International Conference on*, pp. 137–141, IEEE, 2009.

[17] D. Thißen, J. M. E. Carlín, and R. Herpertz, "Evaluating the Performance of an IMS/NGN Deployment," in *GI Jahrestagung*, pp. 2561–2573, 2009.

[18] W. A. Aziz, S. H. El-Ramly, and M. M. Ibrahim, "IP–Multimedia Subsystem (IMS) performance evaluation and benchmarking," in *Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010 IEEE Region 8 International Conference on*, pp. 209–214, IEEE, 2010.

[19] O. University, "Protos security testing of protocol implementations." https://www.ee.oulu.fi/research/ouspg/PROTOS_Test-Suite_c07-sip.

[20] S. Pandey, V. Jain, D. Das, V. Planat, and R. Periannan, "Performance study of IMS signaling plane," in *IP Multimedia Subsystem Architecture and Applications, 2007 International Conference on*, pp. 1–5, IEEE, 2007.

[21] L. Nagy, J. Tombal, and V. Novotny, "Proposal of a queueing model for simulation of advanced telecommunication services over IMS architecture," in *Telecommunications and Signal Processing (TSP), 2013 36th International Conference on*, pp. 326–330, IEEE, 2013.

[22] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 248–260, 1975.

[23] "SIP Express Router." http://www.iptel.org/ser/features/.

[24] "myMONSTER SIP client." http://www.monster-the-client.org/.

[25] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[26] “BIND Manual Pages.” http://www.bind9.net/manuals.