ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

A NOVEL PRIORITY BASED REQUEST SCHEDULING MECHANISM TO PREVENT SIP SERVER OVERLOAD

Ph.D. THESIS

Demir Yaşa YAVAŞ

Electronics and Communications Engineering Department

Telecommunications Engineering Programme

MARCH 2017

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

A NOVEL PRIORITY BASED REQUEST SCHEDULING MECHANISM TO PREVENT SIP SERVER OVERLOAD

Ph.D. THESIS

Demir Yaşa YAVAŞ
(504942017)

Electronics and Communications Engineering Department

Telecommunications Engineering Programme

Thesis Advisor: Prof. Dr. Bilge Günsel KALYONCU
Co-advisor: Dr. İbrahim HÖKELEK

MARCH 2017

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

SIP SUNUCULARIN AŞIRI YÜKLENMESİNİ ÖNLEYİCİ YENİ BİR ÖNCELİKLENDİRME TABANLI İSTEK ZAMANLAMA MEKANİZMASI

DOKTORA TEZİ

Demir Yaşa YAVAŞ
(504942017)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Telekomünikasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Bilge Günsel KALYONCU
Eş Danışman: Dr. İbrahim HÖKELEK

MART 2017

Demir Yaşa YAVAŞ, a Ph.D. student of ITU Graduate School of Science Engineering and Technology 504942017 successfully defended the thesis entitled "A NOVEL PRIORITY BASED REQUEST SCHEDULING MECHANISM TO PREVENT SIP SERVER OVERLOAD", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Bilge Günsel KALYONCU** Istanbul Technical University	.....
Co-advisor :	**Dr. İbrahim HÖKELEK** TÜBİTAK BİLGEM & Istanbul Technical University	.....
Jury Members :	**Doç. Dr. Güneş Zeynep KARABULUT** Istanbul Technical University	.....
	Yrd. Doç. Dr. Ertuğrul BAŞAR Istanbul Technical University	.....
	Yrd. Doç. Dr. Ali Özer ERCAN Özyeğin University	.....
	Prof. Dr. Hakan Ali Çırpan Istanbul Technical University	.....
	Doç.Dr. Öznur Özkasap Koç University	.....

Date of Submission : **17 January 2017**
Date of Defense : **15 March 2017**

To my spouse and children,

FOREWORD

Although, a Ph.D. thesis work seems to be an individual endeavor, it's the teamwork and encouragement that brings it to fruition. I wish to thank my advisor, Prof. Bilge Günsel Kalyoncu, for her continuous guidance, fruitful feedback and moral support. I would also like to thank my co-advisor Dr. İbrahim Hökelek for providing insightful, detailed and constructive comments. I would also like to thank my committee members, Assoc. Prof. Güneş Zeynep Karabulut, Asst. Prof. Ertuğrul Başar and Asst. Prof. Ali Özer Ercan for their insightful comments to include new perspectives to the subject.

I would like to thank Netas for their full support both in financial and equipment terms. Very special thanks to my workfellows in Netas. I also remain indebted for their understanding and support during the times when my performance was down in projects, because of my thesis work.

I am indebted to my wife and daughters for their patience and insightfulness, and especially my wife for her constant and unconditional support.

March 2017 Demir Yaşa YAVAŞ

TABLE OF CONTENTS

Page

FOREWORD ..... **ix**
TABLE OF CONTENTS ..... **xi**
ABBREVIATIONS ..... **xv**
SYMBOLS ..... **xvii**
LIST OF TABLES ..... **xix**
LIST OF FIGURES ..... **xxi**
SUMMARY ..... **xxvii**
ÖZET ..... **xxix**
1. INTRODUCTION **1**
1.1 Main Contributions 4
1.2 Organization of the Thesis 8
2. SIP BACKGROUND **11**
2.1 SIP Protocol Structure 13
2.2 SIP Transactions 14
2.3 SIP Message Structure 17
2.4 SIP Retransmission Mechanism 20
2.5 SIP Retransmission Detection with Transaction Matching 22
3. PRIORITY BASED SIP REQUEST SCHEDULING MECHANISM **25**
3.1 Related Work 27
3.2 The Proposed Priority Scheduling Mechanism 33
3.2.1 Request pre-parsing module 33
3.2.2 Priority queuing module 36
3.2.3 Retransmission removal module 37
4. DISCRETE-TIME FLUID-FLOW MODELS **39**
4.1 Motivation for Fluid-Flow Modeling 40
4.2 Assumptions 41
4.3 Basic Definitions 42
4.4 Arrival Time Slot Tracking Process 44
4.5 Modeling Removal Processes 49
4.5.1 Primary removal process 49
4.5.2 Secondary removal process 50
4.6 Model for Infinite Buffer Case 51
4.6.1 Model for conventional SIP with infinite buffer 51
4.6.1.1 Queuing model 51
4.6.1.2 Modeling retransmissions 52
4.6.1.3 Stability condition 53
4.6.2 Model for PRSM with infinite buffer 54

4.6.2.1 Queuing model 54
4.6.2.2 Modeling retransmissions 55
4.6.2.3 Stability condition 55
4.6.2.4 Retransmission removal processes 56
4.7 Model for Finite Buffer Case 56
4.7.1 Queuing model 56
4.7.2 Modeling retransmissions 58
4.7.3 Removal processes 62
4.7.3.1 Primary removal process 62
4.7.3.2 Secondary removal process 63
4.8 Model for Lossy Network Conditions with Infinite Buffer Case 64
4.8.1 Model for conventional SIP 65
4.8.1.1 Modeling retransmissions 66
4.8.2 Model for PRSM 67
4.8.2.1 Modeling retransmissions 68
4.8.2.2 Removal processes 69
4.9 Performance Metrics 69
4.9.1 Failed call attempts 70
4.9.2 Response delay 71
5. RETRANSMISSION ANALYSIS AND SIMULATION TOOL 75
5.1 MMPP Background 76
5.2 Functional Architecture 78
5.3 Software Architecture 81
5.4 Performance Tests 85
5.4.1 Test results for observer mode 86
5.4.1.1 Results for MMPP(2)/M/1/B 86
5.4.1.2 Results for M/MMPP(2)/1/B 88
5.4.2 Test results for applier mode 89
5.5 Tests for PRSM 94
5.6 Test for Lossy Network Conditions 97
6. PERFORMANCE EVALUATION 109
6.1 Infinite Buffer Case 110
6.1.1 Overload caused by bulk arrivals 110
6.1.1.1 Test duration for 150 seconds 114
6.1.2 Overload caused by temporal server slow down 116
6.1.3 Overload caused by bursty arrivals 120
6.1.4 PRSM stability condition test 123
6.2 Finite Buffer Case 128
6.2.1 Overload caused by bulk arrivals 128
6.2.2 Overload caused by temporal server slow down 131
6.2.3 Overload caused by bursty arrivals 134
6.3 Infinite Buffer Case in Lossy Network Conditions 136
6.3.1 Overload caused by bulk arrivals 137
6.3.2 Overload caused by temporal server slow down 143
6.3.3 Overload caused by bursty arrivals 147

7. A REAL-TIME IMPLEMENTATION WITH JAIN-SIP STACK 153
7.1 JAIN-SIP Background 154
7.2 SIP Server Implementation with PRSM 156
7.3 SIP Client Implementation Sending Retransmission Information 159
7.4 Traffic Performance Tests 160
7.4.1 Modifications on SIPp software 161
7.4.2 Traffic test setup and results 162
8. CONCLUSIONS AND FUTURE WORK 167
8.1 Future Work 168
REFERENCES 169
APPENDICES 175
APPENDIX A.1: Derivation of Conventional SIP Server Model for Infinite Buffer Case 177
1.1 Modeling retransmissions 178
APPENDIX B.1: M/MMPP(2)/1/N Queuing Model 179
CURRICULUM VITAE 182

ABBREVIATIONS

3GPP : 3rd Generation Partnership Project
App : Appendix
ABNF : Augmented Backus-Naur Form
ACK : Acknowledgement
CSV : Comma-Separated Values
ETSI : European Telecommunications Standards Institute
FEFM : Front-End SIP Flow Management system
FIFO : First-In-First-Out
HTTP : HyperText Transfer Protocol
ITU-T : International Telecommunications Union Telecommunication Standardization Sector
IETF : Internet Engineering Task Force
IMS : IP Multimedia Subsystem
IP : Internet Protocol
JAIN : Java APIs for Integrated Networks
MMPP : Markov Modulated Poison Process
NGN : Next Generation Networks
PBX : Private Branch Exchange
PRSM : Priority-based Request Scheduling Mechanism
PSTN : Public Switched Telephone Network
R^3 : Redundant Request Remover
RTP : Real-time Transport Protocol
RTT : Round-Trip Time
SCTP : Stream Control Transmission Protocol
SDP : Session Description Protocol
SIP : Session Initiation Protocol
SOC : SIP Overload Control
SOE : SIP Offload Engine
SRTP : Secure Real-time Transport Protocol
TCP : Transmission Control Protocol
TLS : Transport Layer Security
TU : Transaction User
UA : User Agent
UAC : User Agent Client
UAS : User Agent Server
UDP : User Datagram Protocol
URI : Uniform Resource Identifier
XML : eXtensible Markup Language

SYMBOLS

δ_i	:	MMPP mode duration parameter
B	:	Buffer size
D_n	:	Total delay at time slot n
D_n^p	:	Delay from primary queue processing at time slot n
D_n^s	:	Delay from secondary queue processing at time slot n
λ	:	Original request arrival rate
$\lambda(n)$	:	Original request arrival rate at time slot n
$\lambda^e(n)$	:	Effective original request arrivals that entering the server at time slot n
$\lambda^d(n)$	:	Dropped original request arrival rate due to blocking at time slot n
$\lambda^l(n)$	:	Lost original request arrival rate at time slot n
$l(n)$	:	Loss rate at time slot n
$L(n-k)$	:	Aggregation of arrivals arrived at time slot $n-k$ and processed at time slot n
μ	:	Service rate
$\mu(n)$	:	Service capacity at time slot n
$M(n)$	:	Generic service capacity may represent $\mu(n)$ or $y(n)$ at time slot n
n	:	Current time slot
π	:	Steady-state probabilities
P_b	:	Blocking probability
P_{rt}	:	Retransmission probability
$q(n)$	:	Total queue size at time slot n
$q_p(n)$	:	Primary queue size at time slot n
$q_s(n)$	:	Secondary queue size at time slot n
r	:	Retransmitted request arrival rate
$r(n)$	:	Retransmitted request arrival rate at time slot n
$r^e(n)$	:	Effective retransmitted request arrivals that entering the server at time slot n
$r^d(n)$	:	Dropped retransmitted request arrival rate due to blocking at time slot n
$r^l(n)$	:	Lost retransmitted request arrival rate at time slot n
$r_j(n)$	:	j^{th}-time retransmissions at time slot n
ρ	:	Utilization
S_n	:	Set of processed arrivals at time slot n
T_1	:	SIP timer value for the first retransmission
T_B	:	SIP timer value for the transaction timeout
T_j	:	SIP timer value for the j^{th}-time retransmission
W_n	:	Total number of removals at time slot n
W_n^p	:	Number of removals by primary removal process at time slot n
W_n^s	:	Number of removals by secondary removal process at time slot n
$y(n)$	:	Remaining service capacity after primary queue processing

LIST OF TABLES

Page

Table 5.1 : Comparison between theoretical evaluates and tool results for MMPP(2)/M/1/600. 87
Table 5.2 : Comparison between theoretical evaluates and tool results for M/MMPP(2)/1/600. 89
Table 5.3 : Observed effects of retransmission at queue model with B=1000 and for mode parameters $\delta_0(9) = 0.001953$ and $\delta_1(9) = 0.019531$. 92
Table 5.4 : Number of lost messages compared with the number of arrived messages for 10% loss rate. 98
Table 6.1 : Call failure ratio comparison with various λ_0 values. 124
Table 6.2 : Average arrival rates for selected λ_0 values. 128
Table 7.1 : Reported mean and standard deviation. 164

LIST OF FIGURES

Page

Figure 2.1 : The message flow for SIP voice call session. 12
Figure 2.2 : An example of client and server transactions through SIP elements (based on Figure 4 in RFC 3261). 14
Figure 2.3 : INVITE client transaction finite state machine (RFC 3261). 15
Figure 2.4 : INVITE server transaction finite state machine (RFC 3261). 16
Figure 2.5 : INVITE transaction retransmission message flow. 21
Figure 3.1 : SIP server topologies discussed in RFC 6357. 31
Figure 3.2 : (a) High level functional view of a typical SIP implementation up to Transaction Layer, (b) Its extension with the PRSM. 34
Figure 4.1 : An illustration of time slot tracking process for the primary queue. . 45
Figure 4.2 : Illustration of cases of arrival time slot tracking process. 48
Figure 4.3 : Queuing model for the conventional SIP with infinite buffer. 51
Figure 4.4 : The strict priority queuing model for the PRSM with infinite buffer. 54
Figure 4.5 : The strict priority queuing model of the SIP server with PRSM for finite buffer case. 57
Figure 4.6 : Characterization of the retransmission messages. 59
Figure 4.7 : An illustration of the enqueuing priority assumption for finite buffer case. 60
Figure 4.8 : Queuing model for the conventional SIP with infinite buffer under lossy network conditions. 65
Figure 4.9 : The strict priority queuing model for the PRSM with infinite buffer under lossy network conditions. 68
Figure 5.1 : State transition diagram of MMPP(2)/M/1/B queue. 77
Figure 5.2 : Analysis and Simulation Tool Functional Architecture. 79
Figure 5.3 : Class diagram. 82
Figure 5.4 : Queue sizes that first retransmission occurs for MMPP(2)/M/1/B.... 87
Figure 5.5 : Blocking and retransmission probabilities for MMPP(2)/M/1/B. 88
Figure 5.6 : Queue sizes that first retransmission occurs for M/MMPP(2)/1/B.... 90
Figure 5.7 : Blocking and retransmission probabilities for M/MMPP(2)/1/B. 90
Figure 5.8 : The effects of the retransmissions on the arrival rate for MMPP(2)/M/1/1000. 91
Figure 5.9 : The effects of the retransmissions on the arrival rate for M/MMPP(2)/1/1000. 92
Figure 5.10 : The ratio of the number of retransmissions to the number of the original arrivals for MMPP(2)/M/1/B. 93
Figure 5.11 : The ratio of the number of retransmissions to the number of the original arrivals for M/MMPP(2)/1/B. 94
Figure 5.12 : The ratio of retransmissions to total arrivals versus B. 95

Figure 5.13: The ratio of retransmissions to total arrivals versus B..... 96
Figure 5.14: Mean waiting time for 100-Trying response versus B. 97
Figure 5.15: Queue size behavior of the conventional SIP server with various loss rates..... 99
Figure 5.16: Retransmission rate behavior of the conventional SIP server with various loss rates. 99
Figure 5.17: Call failure ratio for the conventional SIP with various loss rates. ... 100
Figure 5.18: Average delay for the conventional SIP according to various loss rates. 101
Figure 5.19: Delay for individual calls in sequence and the running average delay for the conventional SIP for no-loss case. 101
Figure 5.20: Delay for individual calls in sequence and the running average delay for the conventional SIP for loss rate 10%. 102
Figure 5.21: Total queue size behavior of the PRSM with various loss rates. 103
Figure 5.22: Retransmission rate behavior of the PRSM with various loss rates. . 103
Figure 5.23: Call failure ratio for the PRSM according to various loss rates. 104
Figure 5.24: Average delay for the PRSM according to various loss rates. 104
Figure 5.25: Delay for individual calls in sequence and the running average delay for the PRSM for no-loss case..... 105
Figure 5.26: Delay for individual calls in sequence and the running average delay for the PRSM for loss rate 10%. 105
Figure 5.27: Delay for individual calls in sequence and the running average delay for the PRSM for loss rates (a) 20% and (b) 30%, respectively. 106
Figure 5.28: Total queue size behavior of the PRSM with removal deactivated for various loss rates. 106
Figure 5.29: (a) Failed call attempt ratio for the PRSM with no removal according to various loss rates, (b) Average delay for the PRSM with no removal according to various loss rates. 107
Figure 5.30: Delay for individual calls in sequence and the running average delay for the PRSM with no removal for loss rate 10%. 107
Figure 6.1 : The queue size for the overloaded conventional SIP server when $q(0)$ violates the stability condition. 110
Figure 6.2 : The retransmission rate for the overloaded conventional SIP server when $q(0)$ violates the stability condition..... 111
Figure 6.3 : The primary and secondary queue sizes for the overloaded SIP server with the PRSM when $q(0)$ violates the stability condition. 112
Figure 6.4 : The retransmission rate for the SIP server with the PRSM when $q(0)$ violates the stability condition. 113
Figure 6.5 : Primary and secondary queue sizes for the SIP server with the PRSM when the removal process is disabled and $q(0)$ violates the stability condition. 114
Figure 6.6 : Results from the extended test case-1 as changes in queuing dynamics (a) queue size, (b) retransmission rate, and performance metrics (c) failed call attempts, (b) response delay, versus time..... 115
Figure 6.7 : Service and original message arrival rates for overloading SIP server performing normal and maintenance mode services alternately. 116

Figure 6.8 : Queue size versus time for the overloaded conventional SIP server which performed normal service and maintenance service alternately. 117
Figure 6.9 : The retransmission rate for the overloaded SIP server which performs normal and maintenance service modes alternately. 118
Figure 6.10 : Primary and secondary queue sizes for the overloaded SIP server with the PRSM, which performs normal and maintenance service modes alternately. 119
Figure 6.11 : The retransmission rate for the overloaded SIP server with the PRSM which performs normal and maintenance service modes alternately. 120
Figure 6.12 : Original message arrival rate and server service rate for pseudo-random mode transitions of MMPP. 121
Figure 6.13 : Change of the retransmission rate for the conventional SIP server overloaded by bursty arrivals (fluid-flow model). 122
Figure 6.14 : Change of the queue size for the conventional SIP server overloaded by bursty arrivals (fluid-flow model). 123
Figure 6.15 : Primary and secondary queue sizes for the server with the PRSM which is overloaded by bursty arrivals. 124
Figure 6.16 : Queue size versus time for λ_0 settings 995, 1045, 1095, 1145 and 1195 125
Figure 6.17 : Retransmission rate versus time for λ_0 settings 995, 1045, 1095, 1145 and 1195. 126
Figure 6.18 : Response delay versus time for λ_0 settings 995, 1045, 1095, 1145 and 1195. 127
Figure 6.19 : Queue size change for the conventional SIP server when the initial queue size is set to 6K messages, for various buffer sizes, B. 128
Figure 6.20 : Change of the retransmission rate for the conventional SIP server and for the PRSM when the initial queue size is set to 6K messages, for various buffer sizes, B. 129
Figure 6.21 : PRSM queue size change for buffer sizes, 6K, 10K and 20K for fluid-flow model and simulation. 130
Figure 6.22 : Queue size change for the overloaded SIP server which performs normal and maintenance service modes alternately (model and simulation). 132
Figure 6.23 : Change of the retransmission rate for the conventional SIP server which performs normal and maintenance service modes alternately (fluid-flow model). 132
Figure 6.24 : Queue size change for SIP server with PRSM which performs normal and maintenance service modes alternately. 133
Figure 6.25 : Retransmission rate change for SIP server with PRSM which performs normal and maintenance service modes alternately. 133
Figure 6.26 : The mean ratio of the number of failed call attempts to the number of all call attempts for various buffer sizes. 134
Figure 6.27 : The mean delay for sending 100-Trying response for various buffer sizes. 135

Figure 6.28: Queue size change for the conventional SIP server when the initial queue size is set to 6K messages, for various loss rates. 136
Figure 6.29: Change of the retransmission rate for the conventional SIP server when the initial queue size is set to 6K messages, for various loss rates. 137
Figure 6.30: Average response delays for the conventional SIP server when the initial queue size is set to 6K messages, for loss rates between 0%-90%. 138
Figure 6.31: The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the conventional SIP server when the initial queue size is set to 6K messages. 139
Figure 6.32: Delay for each individual call in sequence and the running average delay for the conventional SIP under 10% loss rate. 140
Figure 6.33: Queue size change for the PRSM when the initial queue size is set to 6K messages, for various loss rates. 140
Figure 6.34: Change of the retransmission rate for the PRSM when the initial queue size is set to 6K messages, for various loss rates. 141
Figure 6.35: Average response delays for the PRSM when the initial queue size is set to 6K messages, for loss rates between 0%-90%. 141
Figure 6.36: The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the PRSM when the initial queue size is set to 6K messages. 142
Figure 6.37: Delay for individual calls in sequence and the running average delay for the PRSM under loss rate 10%. 142
Figure 6.38: Queue size change for the conventional SIP server for various loss rates when the server is overloaded by temporal slow down. 143
Figure 6.39: Change of the retransmission rate for the conventional SIP server for various loss rates when the server is overloaded by temporal slow down. 144
Figure 6.40: Average response delays for the conventional SIP server for loss rates between 0%-90%. 145
Figure 6.41: The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the conventional SIP when the server is overloaded by temporal slow down. 145
Figure 6.42: Queue size change for the PRSM when the server is overloaded by temporal slow down, for various loss rates. 146
Figure 6.43: Change of the retransmission rate for the PRSM when the server is overloaded by temporal slow down for loss rates (a) 10%, (b) 30%, (c) 50% and (d) 70%..... 147
Figure 6.44: Average response delays for the conventional SIP server for loss rates between 0%-90%. 148
Figure 6.45: The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the PRSM when the server is overloaded by temporal slow down. 148

Figure 6.46: Queue size change for the conventional SIP server under bursty traffic conditions, for various loss rates. 149
Figure 6.47: Change of the retransmission rate for the conventional SIP server under bursty traffic conditions, for various loss rates. 149
Figure 6.48: Average response delays for the conventional SIP server for loss rates between 0%-90%. 150
Figure 6.49: The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the conventional SIP server. 151
Figure 6.50: Average response delays for the PRSM server for loss rates between 0%-90%. 151
Figure 6.51: The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the PRSM. 152
Figure 7.1 : The Architecture of JAIN-SIP Stack Implementation. 154
Figure 7.2 : (a) High level functional view of a typical SIP Implementation up to Transaction Layer, (b) Its extension with Request Prioritization Mechanism)..... 156
Figure 7.3 : An example of INVITE message in a SIPp scenario file as including "rb" parameters. 161
Figure 7.4 : The call scenario set for traffic tests. 162
Figure 7.5 : The response delays measured in real-time experiments. 163
Figure 7.6 : Operational measurements of PRSM reported for every 1 minute.... 164
Figure A.1 : Queuing model for the conventional SIP with finite buffer. 177
Figure B.1 : M/MMPP/1/B Queue State Transition Diagram. 179

A NOVEL PRIORITY BASED REQUEST SCHEDULING MECHANISM TO PREVENT SIP SERVER OVERLOAD

SUMMARY

As being one of core enabling technologies for interoperable unified communications, Session Initiation Protocol (SIP) represents the next step in the evolution of Internet technologies within both public Internet and corporate intranets. With the capability of enabling multimedia communications such as voice over IP (VoIP), instant messaging (IM), telephony presence information as well as video conferencing to be in a single network session, it provides an "unified" communication for enterprise networks. SIP has been adopted by 3rd Generation Partnership Project (3GPP), International Telecommunications Union Telecommunication Standardization Sector (ITU-T), and European Telecommunications Standards Institute (ETSI) to IP Multimedia Subsystem (IMS) in next generation networks (NGN) as a core signaling protocol to provide voice services for mobile communication over IP. With the increasingly wide deployment, SIP encountered scalability issues and server overload problems leading to severe performance degradation similar to the cases with the previous signaling protocols used in telecommunication.

The recommended overload control mechanism of SIP, which is based on the 503 (Service Unavailable) response code, does not prevent the congestion collapse, and may even cause traffic to oscillate between servers and spread the overload condition throughout the network. As an application layer signaling protocol, SIP can be transported over Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Stream Control Transmission Protocol (SCTP). While the retransmission mechanism is introduced to increase the reliability when SIP is transported over UDP, the overload condition is exacerbated by the regenerative behavior of the retransmission mechanism. In this case, the offered load for the already overloaded server is significantly increased by triggering redundant retransmissions. This situation causes more messages to be delayed for longer duration, which may eventually results in a congestion collapse and a server crash.

In this thesis, we propose a novel priority-based SIP request scheduling mechanism (PRSM) to prevent server overloading and to improve the scalability of SIP servers by diminishing the negative impact of redundant retransmissions. In the proposed system, the original incoming SIP requests have strict priority over the retransmitted requests that limit the number of the transmissions by preferring to serve the original incoming requests first. Note that parsing of text based SIP messages is a time consuming task, and it is prohibitively expensive if the identification of retransmissions requires parsing of entire message. Therfore, we introduced a new field carrying a retransmission information into the request-line of the SIP request message to identify the retransmissions with a minimum parsing effort. To further minimize the redundant usage of limited server resources, we introduced a retransmission removal procedure to

erase redundant retransmissions from the server queue when a corresponding message is successfully processed.

Towards characterizing the main factors affecting the PRSM behavior under various overload conditions, we develop discrete-time fluid-flow models representing various server and network conditions, such that the server with infinite and finite buffers, various traffic arrival and departure processes, and lossy network conditions. To verify the accuracy of the proposed models, we design and implement a discrete-event based retransmission analysis and simulation tool which is capable of generating realistic arrival and departure patterns, lossy network behavior and controlling various SIP server parameters with reporting capabilities. The software architecture is designed to support extensions on queuing and traffic models. Finally, we design and develop a proof-of-concept of the PRSM in a real-time SIP stack to verify the mechanism is feasible for implementation and can co-exist with the conventional SIP design in the same stack implementation.

We conducted test cases using the derived fluid-flow models and our retransmission analysis and simulation tool to demonstrate the scalability improvements of the PRSM. The test cases are conducted to create overload conditions in the server under bulk arrivals, server slowdown situations and demand bursts, and consequently, to trigger significantly high retransmissions leading the server crash for the conventional SIP server. Performance evaluations with both the analysis and simulation tool and models indicate that for higher buffer sizes, the PRSM performs significantly better than the conventional SIP. In the higher buffer size case, there will be more redundant retransmissions in the network due to higher queuing delays and the results demonsrated that the priority mechanism of the PRSM can cope with this situation. Consequently, the PRSM provides network administrator with the ability to configure the buffer size of a SIP server to a moderately high value as having significantly and consistently better scalability compared to the conventional SIP.

SIP SUNUCULARIN AŞIRI YÜKLENMESİNİ ÖNLEYİCİ YENİ BİR ÖNCELİKLENDİRME TABANLI İSTEK ZAMANLAMA MEKANİZMASI

ÖZET

Karşıklı çalışabilir birleşik haberleşmeyi olanaklı kılan temel teknoloji olarak, SIP (Session Initiation Protocol), hem geniş kapsamlı genel Internet, hem de şirket-içi şebekelerde, Internet teknolojilerinin gelişimindeki bir sonraki adımı temsil eder. Konuşma, anlık mesaj (instant messaging (IM)), telefon uygunluk durumu bilgisi ve video konferans gibi çoklu-ortam haberleşmelerini bir oturum üzerinde gerçekleştirme olanak ve yeteneği ile birlikte, işletme şebekeleri için "birleşik" bir haberleşme ortamı sağlar. SIP, aynı zamanda 3GPP (3rd Generation Partnership Project), ITU-T (International Telecommunications Union Telecommunication Standardization Sector) ve ETSI (European Telecommunications Standards Institute) tarafından, sonraki nesil şebekelerde (Next Generation Networks (NGN)) yer alan IMS (IP Multimedia Subsystem) için, mobil iletşimi IP üzerinden sağlamak üzere, temel işaretleşme protokolü olarak uyarlanmıştır. Artan geniş çaplı kullanımı, tüm önceki vadesi dolmuş protokollerde olduğu gibi, SIP'i de performans kayıplarına neden olan sunucu aşırı yüklenmesi problemi ile karşı karşıya bırakmıştır. SIP standardı tarafından 503 (Service Unavailable) yanıt koduna bağlı olarak önerilen aşırı yüklenme kontrol mekanizması tıkanıklıktan çökmeyi önleyemediği gibi, aşırı yüklenme probleminin, sunucular arasında gidip gelerek tüm şebekeye yayılmasına neden olabilir.

Bir uygulama katmanı işaretleşme protokolü olarak, SIP, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) ve SCTP (Stream Control Transmission Protocol) üzerinden iletilebilir. Her ne kadar iletimin güvenirliğini arttırmak adına, güvenilir olmayan UDP için istek mesajlarının yeniden iletilmesi mekanizması tanımlanmış olsa da, zaten aşırı yüklenmiş olan sunucuya gereksiz yeniden iletimlerin tetiklenmesine neden olan yeniden iletim mekanizmasının doğurgan yapısı aşırı yüklenme durumunu daha da kötüleştirir. Öyle ki, bu durum sunucunun çökmesine kadar varabilir.

Her ne kadar bilinen SIP tasarımı fazlalık yeniden iletimleri tanımlayabilse de, bunun için SIP istek mesajının tümünün ayrıştılmasına gerek duyulur. Ancak, SIP mesajları metin tabanlıdır ve satırlardan oluşur. Mesajın bir istek ya da yanıt mesajı olduğu ilk satırı ile belirlenir ve bu satırlar özel bir formata sahiptir. Bu satır mesajın tipine göre *"istek satırı"* (request-line) ya da *"durum satırı"* (status-line) olarak adlandırılır. Sonraki satırlar, mesajın ana içeriğine kadar, birer başlık ile birlikte, gönderen, gönderilen ve oturum kimliği gibi bir SIP mesajında olması beklenen parametreleri içeren satırlardan oluşur. Bu satırların mesaja yerleştirilmesinde bir sıralama zorunluluğu olmadığı gibi, her bir başlık altındaki parametrelerin de belirli bir sırada olması beklenmez. Metin tabanlı mesajların okunabilirliği ve esnek yapısı tasarım aşamasında kolaylık sağlasa da, ayrıştırmada alıcı tarafta önemli bir yük oluşturur. Her bir başlık ya da parametrenin tanınması karakter dizilerinin bir bir karşılaştırılmasını gerektirir. Bu durum, SIP mesajlarının esnek yapısı ve parametre

zenginliği ile birleşince, bir SIP masajının işlenmesini oldukça fazla işlem gücü gerektiren bir iş durumuna getirir ki bu tasarım gerçekleme biçimine ve kullanılan programlama diline bağlı olarak, bir mesajın tümüyle işlenmesi için harcanan zamanın %25-%40'ını alır.

Ek olarak, bilinen SIP tasarımı, alınan bir istek mesajının yeniden iletilen bir mesaj olduğunun fark edilmesi durumunda, bu mesaj için en son gönderilmiş olan yanıt mesajının yeniden gönderilmesini önermektedir. Alınan istek mesajının fazlalık olması durumunda, bu davranış, sistem kaynaklarının ve şebeke kapasitesinin verimsiz kullanılması anlamına gelir. Dolayısıyla, bir fazlalık istek mesajına harcanan zaman ve gönderilen yanıt mesajı da fazlalıktır.

Sonuç olarak, bilinen SIP tasarımı, önerdiği yaklaşımla, fazlalık istek mesajlarının yarattığı olumsuz etkiyi karşılayabilecek donanıma sahip değildir. SIP sunucuların aşırı yüklenmesi ile ilgili çalışmalar da bunu göstermektedir. Bu durumdan yola çıkarak, bu tezde, aşırı yüklenmiş durumda da olsa, fazlalık yeniden iletimlerin SIP sunucular üzerindeki olumsuz etkisini azaltacak, sunucunun ölçeklenirliğini (scalability) arttıracak yeni bir yaklaşım olarak, PRSM (Priority-based Request Scheduling Mechanism) olarak adlandırdığımız, önceliklendirme tabanlı SIP istek zamanlama (scheduling) mekanizması önerilmektedir. PRSM'de orijinal istek mesajlarının (başlangıçtaki, ilk iletilen istek mesajı) yeniden iletimlere göre öncelikli olarak kuyruklandırıldığı bir kuyruk yapısı vardır. Mesajların işlenmesi sırasında yeniden iletimler ancak ve ancak, kuyrukta orijinal bir istek mesajı yoksa işlenmesi biçiminde katı kuralcı bir önceliklendirme (strict priority) öngörülmektedir. Buradaki temel amaç, orijinal mesajlara olabidiğince hızlı yanıt verip olası fazlalık yeniden iletimleri engellemektir. Alınan fazlalık yeniden iletimleri, mesajın tümünü ayrıştırmadan ayırt edebilmek için, PRSM'de, SIP istek satırına, yeniden iletim bilgisini ve mesajın kimliğini taşıyan yeni bir parametre önerilmektedir. Böylelikle her bir istek mesajı, tümüyle ayrıştırılmaksızın, önceliklendirme ölçütlerine göre kuyruklanır ve katı kurallı önceliklendirmeye göre işlenip sunucu yazılımının üst katmanlarına verilir. Bu aşamada ya da mesaj üst katmanlarda işlendiğinde bu mesajla ilişkili tüm fazlalık iletimler kuyruktan uzaklaştırılabilir. Mesaj istek satırına eklenen mesaj kimliği bu işlemin üst katmanlara verilmeden de yapılabilmesini sağlar. Böylelilke üst katmanlardaki işlemler bilinen SIP tasarımına göre devam edebilir.

Özetlenirse, PRSM üç temel işlev tanımlamaktadır: *(i)* SIP istek mesajının istek satırını ayrıştıran ön-ayrıştırıcı (pre-parser), *(ii)* katı kuralcı önceliklendirmeli kuyruk yapısı, ve *(iii)* fazlalık yeniden iletimlerin kuyruktan uzaklaştırılması. Bu işlevlerin gerçek bir SIP yığınında (stack) gerçeklenebileceğini sınamak için açık kaynak kodlu JAIN-SIP Stack üzerinde bir ön tasarım ve gerçekleme (proof-of-concept) yapılmıştır. Böylelikle PRSM'in varolan bir SIP yığıtında temel bir değişiklik gerektirmeden gerçeklenebileceği sınanmıştır. Yeniden iletim bilgisi içermeyen istek mesajlarının yüksek öncelikli olduğu varsayılarak, bilinen SIP ve PRSM'i destekleyen SIP istemcilerinin aynı sistemde olabilmesi durumu test edilmiştir.

Bu arada geliştirdiğimiz ayrık-olaylı (discrete-event) bir yeniden iletim analiz ve benzetim (simulation) aracı hem yeniden iletimlerin SIP sunucular üzerindeki etkisini hem de PRSM'in öngördüğü öncelikli kuyruk yapısının davranışını incelemek için kullanılmıştır. Aracın trafik modeli, paket temelli haberleşme şebekelerinde patlamalı (bursty) geliş yoğunluklarını modellemek için sıklıkla kullanılan, MMPP (Markov Modulated Poisson Process) üzerine kurulmuştur. MMPP, aynı zamanda, sistemin

normal çalışma durumunu ve periyodik bakım durumunu yansıtabilecek farklı servis süresi yoğunluklarını modellemek için kullanılmaktadır. Bu amaçla, analiz ve benzetim aracı hem talep patlamalarını hem de sunucunun işlem kapasitesinde düşüşe neden olabilecek durumları aynı anda ve ayrı ayrı modelleyecek biçimde tasarlanmıştır. Ek olarak, analiz ve benzetim aracı, normal ve yeniden iletilen mesajları sunucu kuyruğunda önceliklendirebilecek biçimde yapılandırılabilmektedir. Bu özellik, çeşitli önceliklendirme varsayımları altında analiz yapma olanağı sağlamaktadır. Arabellek büyüklüğü (buffer size) sınırlandırılabilmekte, böylelikle sonlu ve sonsuz arabellek durumları incelenebilmektedir. Araç şebeke kayıplarının benzetimine de olanak vermektedir. Böylelikle, tümüyle fazlalık yeniden iletim üreten sonsuz bellek durumu ve kuyrukta bloklanan mesajlardan dolayı ve şebekedeki kayıplardan dolayı fazlalık olmayan yeniden iletim üreten, sırasıyla, sonlu arabellek ve şebeke mesaj kaybı olması durumlarının incelenmesine olanak sağlanmıştır. Aracın doğruluğu MMPP trafik ve servis süresi modellerinin teorik beklentileri ile karşılaştırmalar yaparak sınanmıştır. Bu araç, PRSM için geliştirdiğimiz, ayrık-zamanlı sıvı-akış (discrete-time fluid-flow) temelli modelleri sınamak için de kullanılmıştır.

Yeniden iletimin doğurgan yapısı, dolayısıyla, özellikle kuyruk gecikmelerinin neden olduğu fazlalık yeniden iletimlerin ve bu iletimlere maruz kalan bir sistemin, durağan sistemleri modellemekte kullanılan analitik yöntemlerle modellenememesi nedeniyle, PRSM'i modellemek için ayrık-zamanlı sıvı-akış temeli modelleme kullanılmıştır. Ayrık-zamanlı sıvı-akış temelli modellemede zaman sabit zaman dilimlerine bölünmekte ve kuyruk uzunluğu gibi kuyruk dinamiklerinin bir sonraki zaman dilimi için değeri farksal denklemlerle hesaplanmaktadır. Her zaman diliminde, SIP yeniden iletim mekanizmasına uygun olarak, zamanında işlenememiş ya da önceki iletiminde bloklanmış ya da şebekede kaybolmuş istek mesajlarının yeniden iletilme durumu hesaplanıp, sisteme gelen yeniden iletimiş mesajlar hesaplanmaktadır. İçinde bulunulan zaman dilimi için belirli bir dağılıma göre gelen orijinal mesajlarla, hesaplanan yeniden iletilen mesajlar sisteme o zaman dilimi için gelen trafiği oluşturmaktadır. Sunucu sistemde bulunan (o zaman diliminde ya da önceden gelip kuyruğa girmiş) istek mesajlarının bir miktarını işleyeceğinden, kuyrukta kalan kısım, bir sonraki zaman dilimindeki kuyruk uzunluğunu vermektedir. Orijinal mesajların geliş hızı dağılımı, sunucunun mesaj işleme kapasitesi ve modelin tanımına göre, arabellek büyüklüğü ve/veya şebeke kayıp oranı modelin girdileri olup, yeniden iletim ve kuyruk dinamiklerinin özyinelemeli bir biçimde hesaplanması gerekmektedir. Bu modelleme yaklaşımında orijinal mesajların geliş hızı ve sunucunun işleme kapasitesi dağılımları gelişigüzel seçilebilmektedir.

Bu tezde PRSM'i modelleyen üç ayrık-zamanlı sıvı-akış modeli türetilmiştir: *(i)* sunucunun sonsuz arabelleğe sahip olduğu durum, *(ii)* sunucunun sonlu arabelleğe sahip olduğu durum, ve *(iii)* sunucunun sonsuz arabelleğe sahip olduğu, ancak şebeke kayıplarının olduğu durum. Tüm model durumlarında tek sunuculu kuyruk modeli varsayılmıştır. Literatürde sonsuz ve sonlu arabellek durumları için bilinen SIP sunucuları için ayrık-zamanlı sıvı-akış modelleri bulunmaktadır. Ancak, durum *(iii)* için model bu tez çalışmasında türetilmiştir. Bilinen SIP için türetilmiş modeller de programlanarak PRSM'in performansı ile karsılaştırmalar yapılmıştır. Temel olarak, tüm modellerde, iletim ortamındaki gecikmenin önemsenmeyecek kadar küçük olduğu, fazlalık yeniden iletimlerde kuyruk gecikmesinin etkin olduğu varsayılmıştır.

PRSM'in öncelikli kuyruklama kullanmasının yanı sıra, bilinen SIP tasarımından temel farkı, bir mesaj işlendiğinde ilişkili tüm fazlalık yeniden iletimlerin kuyruktan uzaklaştırılmasıdır. Bu davranışın modellenebilmesi için, bulunulan zaman diliminde işlenen bir mesajın aslında hangi zaman diliminde sunucuya geldiğinin bilinmesi gerekir. Bu bilgi kullanılarak, işlenen mesajın ne kadar geciktiği, dolayısıyla bu mesaj için kaç yeniden iletim yapılmış olabileceği bulunabilir. Bu amaçla, *"varış zaman dilimi izleme süreci"* (arrival time slot tracking process) isimli yeni bir süreç tanımlanmıştır. Dolayısıyla, yeni tanımlanan *"uzaklaştırma süreci"* (removal process) ilişkili yeniden iletimleri *"varış zaman dilimi izleme süreci"*'den yararlanarak bulabilmektedir.

Bilinen SIP modelleri için de geçerli olmak üzere, başarısız çağrı miktarı/oranı ve yanıt gecikmesi olarak, iki performans ölçütü tanımlanmış ve ilgili formüller türetilmiştir. Başarısız çağrıların hesaplanmasında, SIP standartı ile uyumlu olarak, bir istek mesajına 32 saniye hiç bir yanıt alınamaması tanımı kullanılmıştır. Yanıt gecikmesi ise istek mesajının sisteme gelme zamanı ile bir yanıt gönderilmesi arasındaki süre olarak tanımlanmıştır. Yanıt gecikmesinin hesaplanması *"varış zaman dilimi izleme süreci"*nin kullanılmasını gerektirmektedir.

Ayrık-zamanlı sıvı-akış modellerinin performans değerlendirmesi aşırı yüklenmelerin *(i)* sunucuya başlangıçta belirli bir mesaj yığının gelmiş olması, *(ii)* sunucunun belirli zaman aralıkları için bakım moduna girmesi ve *(iii)* sunucunun MMPP trafik modeli ile oluşturulan bir patlamalı (bursty) trafik ile karşı karşıya kalması ile oluştuğu üç test durumu öngörülerek yapılmıştır. Testlerde kuyruk uzunluğunun ve yeniden iletim trafiğinin zamanla değişimi için sonuçlar oluşturulmuştur. Modellerin doğruluğu yeniden iletim analiz ve benzetim aracımızdan alınan sonuçlarla sınanmış ve oldukça yakın sonuçlar içerdikleri gözlenmiştir. Tüm testler bilinen SIP için de koşturulmuş ve PRSM için elde sonuçlarla karşılaştırılmıştır. Performans karşılaştırmaları gösteriyor ki, ilk yeniden iletim zamanından (SIP standardı ilk yeniden iletim zamanını 500 ms olarak önermektedir) daha küçük gecikmeler yaratacak düzeyde küçük arabellek büyüklüğü durumunda, başarısız çağrı oranları ve yanıt gecikmeleri açısından, bilinen SIP daha iyi sonuçlar vermektedir. Ancak daha büyük arabellek değerlerinde PRSM çok daha iyi sonuçlar vermektedir. Örneğin bilinen SIP için yanıt gecikmesi 179 saniyeleri bulabilirken, PRSM'de 175 ms düzeyinde olmaktadır. Trafik yoğunluğnun çok yüksek seçildiği testlerde ve sonsuz arabellek varsayımı altında, analiz aracı bellek yetersizliğinden çökerken, PRSM testi sorunsuz tamamlayabilmektedir. Bunun yanında, not etmeliyiz ki, küçük arabellek seçimi bir aşırı yüklenme önleyiçi olarak kullanılabilir ancak bir anlamda gereksiz yere üretilmesine neden olunan fazlalık olmayan trafik şebekenin kapasitesinin verimsiz bir biçimde kullanılmasına neden olabileceği göz önünde bulundurulmalıdır. Genel olarak, PRSM'in operatörlerin SIP sunucuları yüksek arabellek ayarlarıyla çalıştırabilmesine olanak verdiğini söylemek mümküdür. Ancak temelde söyleyebileceğimiz, PRSM'in SIP sunucu performansını iyileştirdiği ve ölçeklenirliğini (scalability) arttırdığıdır, yani aynı sistem kapasitesi ile daha fazla çağrı işlenebilmektedir.

1. INTRODUCTION

Session Initiation Protocol (SIP) [1] has been widely used application layer signaling protocol for establishing, modifying, and terminating multimedia sessions in Internet and Internet Protocol (IP) telephony applications such as Voice-over-IP (VoIP), instant messaging, event notification, and video conferencing. SIP has been standardized by Internet Engineering Task Force (IETF) and was adopted by 3rd Generation Partnership Project (3GPP), International Telecommunications Union Telecommunication Standardization Sector (ITU-T) and European Telecommunications Standards Institute (ETSI) as the core signaling protocol for IP Multimedia Subsystem (IMS) in next generation networks (NGN).

With the increasing usage of SIP servers in large deployments, it is observed that the existing SIP design does not scale up for large network sizes and SIP servers are not well equipped to handle overload situation. The main reasons of SIP server overload are identified as poor capacity planning, component failures, avalanche restart, flash crowds, and denial of service attacks [2]. An overload condition causes performance degradation that may result in a server crash leading the failure of the entire system. The SIP basic overload control mechanism defined in RFC 3261 [1], which uses 503 (Service Unavailable) response code and Retry-After header, cannot fully prevent the overload of a SIP server and congestion collapse. In fact, its on/off behavior may cause traffic to oscillate between SIP servers and thereby worsen an overload condition [2].

IETF SIP Overload Control (SOC) working group published a design consideration document RFC 6357 [3] that presents a SIP overload control mechanism using the requirements given in RFC 5390 [2]. Specifically, local and distributed SIP overload control mechanisms are discussed. The local overload control is an internal mechanism of a server and used to keep the performance of the server at an acceptable level under overload conditions. The distributed overload control mechanism can work explicitly using feedback from downstream servers or implicitly using a timeout mechanism. The distributed overload control mechanism can be used in hop-by-hop or end-to-end

modes. However, as stated in RFC 6357, since the overload control is based on reducing the number of messages a sender is allowed to send, it is only applicable for server-to-server topologies and not suited for client-to-server topologies, named also as *"edge proxy"* topology.

The SIP standard defines an application layer retransmission mechanism for the unreliable User Datagram Protocol (UDP), such that if the response has not been received for a timeout duration, the request is retransmitted by doubling the timeout duration in each iteration. This regenerative nature of the retransmission mechanism increases likelihood of the server overloading [2, 3]. For example, when the number of SIP messages including retransmissions exceeds the processing capacity of the server, the queue size increases and eventually the SIP server can be overloaded. This results in a high queuing delay and trigger further redundant retransmissions which may eventually cause the congestion collapse of the system.

In this thesis, we propose a novel priority-based SIP server request scheduling mechanism (PRSM) to increase the scalability of SIP servers by diminishing the negative effect of redundant retransmissions. The main idea is to identify retransmitted messages before queuing and to provide strict priority for the original incoming SIP requests over the retransmissions. For the message identification purpose, the server has to parse a minimal amount of the SIP message, since its parsing is a time consuming process [4–6]. To achieve this objective, the PRSM defines a parameter in the SIP request line consists of the retransmission sequence number and the request identification. This parameter is used to identify retransmissions with a minimal message parsing effort. This identification is necessary to place an incoming request to its corresponding priority queue such that a new request to the high priority (primary) queue and a retransmission request to the low priority (secondary) queue. When a request message is started to be served, the corresponding redundant retransmissions are removed from the low priority queue.

The modeling retransmission traffic is a challenging task since the retransmission mechanism generates duplicate messages, which have a complex correlation structure due to various timers [7], such that the retransmission mechanism amplifies the SIP overload and makes an overloaded SIP server unstable, existing models created for a stable SIP system cannot be effectively used to analyze the dynamic performance

of an overloaded server. As based on this fact, we derived discrete-time fluid-flow models towards identifying the main factors defining the PRSM behavior under various overload conditions. We consider three models consisting of infinite and finite buffer cases and lossy network conditions (only for infinite buffer case). Models for the first two are based on the models on [8, 9], while the last is inspired by the approach used for the finite case.

As different from the models for the conventional SIP server, the models for the PRSM utilizes primary and secondary queues for the original request messages and the retransmitted requests, respectively. Queues are scheduled according to the non-preemptive strict priority fashion. The *arrival time slot tracking process* is proposed to keep track of time slots of request messages when they arrived into the system and are picked for service processing. The *removal process* is proposed to remove redundant retransmissions from queue upon the corresponding request message is served by using the information from the *arrival time slot tracking process*. The performance metric namely the failed call attempts and the response delay from sending INVITE request until receiving a 100-Trying response are derived using the equations of proposed fluid-flow models.

Meanwhile, we developed an exploratory discrete-event based retransmission analysis and simulation tool to fully understand the behavior of SIP server under various traffic loadings with the capability of generating realistic arrival and departure patterns and varying SIP server queuing parameters. We consider that this tool can be used to evaluate the performance of SIP server under both demand burst and diversity on the server processing capability. For example, the avalanche restart phenomena described in RFC 5390, which causing a flood of SIP REGISTER messages while the server needs to access a database for user credential verification, can be modeled. Therefore, Markov Modulated Poisson Process (MMPP) is used in the tool for generating various arrival and departure patterns corresponding to both cases. With the capability of switch-off retransmission mechanism, it can be used to observe queuing dynamics causing retransmissions. We tested the accuracy of the tool with theoretical expectations of MMPP arrival and departure models.

The performance evaluation of fluid flow models are performed numerically by implementing the fluid-flow equations in Java and the accuracy of the model is

validated by the discrete-event based retransmission analysis and simulation tool. We conducted test cases under the heavy traffic conditions, where the overload is caused by bulk and bursty arrivals or server slowdown. For the infinite buffer case, while the conventional SIP server entered instable situation with continuously increasing queue size, which can lead a server crash, the PRSM coped with overload conditions for all test cases. For the finite buffer case, by generating realistic traffic using the MMPP arrival model, the experiments demonstrate that the number of failed call attempts of the PRSM is reduced to zero when the buffer size is higher than 1K while it linearly increases for the conventional SIP. Furthermore, the mean response delay from sending an INVITE request until receiving a 100-Trying response is kept constant around 175 ms using the PRSM, while it is significantly higher for the conventional SIP. The numerical results not only verify the effectiveness of the PRSM but also closely match with the simulation results for all experiments, indicating that the derived fluid flow formulation can accurately capture the dynamic behavior of the PRSM.

We designed and implemented a proof-of-concept of the PRSM in a real-time system, JAIN-SIP stack, to demonstrate that its implementation in real systems is feasible and requires minimal changes. Also the JAIN-SIP stack implementation is tested under real-time traffic provided by SIPp traffic generator. The preliminary results of the JAIN-SIP stack implementation indicates that the PRSM provides consistently better scalability at the SIP server.

1.1 Main Contributions

In this thesis, we have the following four main contributions:

1. We proposed the novel priority-based SIP request scheduling mechanism (PRSM) in Chapter 3 to increase the scalability of SIP servers by diminishing the negative effect of redundant retransmissions. A new extension to the SIP request-line is defined to identify retransmissions with the minimal amount of processing power. New functional blocks, namely *Pre-parser Module*, *Strict Priority Queue* and *Retransmission Removal Module*, are defined as high-level blocks of a typical SIP server implementation up to SIP Transaction Layer. Test cases are conducted to evaluate the performance of the PRSM using both our retransmission analysis

and simulation tool and discrete time fluid-flow models. Using the analysis and simulation tool and configuring the SIP server with high buffer sizes, we conducted test cases with various traffic loadings creating significantly higher amount of retransmissions and eventually resulting in the server crash for the conventional SIP server. It is verified that the PRSM can cope with the overload conditions in these settings demonstrating significant scalability improvements. The following test cases are conducted for discrete-time fluid-flow modeling cases to evaluate the effect of the PRSM on the SIP server scalability:

(a) overload caused by bulk arrivals, where the server has initially arrived original requests,

(b) overload caused by server slowdown, where the server enters maintenance mode for various times,

(c) overload caused by bursty arrivals, which is based on MMPP arrival model.

The mechanism is also verified in terms of applicability to real-time SIP stacks with a proof-of-concept implementation on JAIN-SIP stack. The preliminary real-time traffic test results of the JAIN-SIP stack implementation obtained with SIPp traffic tools also indicate that the PRSM provides consistently better scalability at the SIP server.

2. We derived discrete-time fluid-flow models for the PRSM in Chapter 4 towards characterizing the main factors affecting the PRSM behavior under various overload conditions. The following modeling cases are considered:

   (a) the PRSM with infinite buffer,

   (b) the PRSM with finite buffer,

   (c) the conventional SIP server and the PRSM with infinite buffer in the lossy network conditions.

   The *arrival time slot tracking* is proposed to keep track of time slots of request messages when they arrived into the system and are picked for service processing. The *removal process* is proposed to remove redundant retransmissions from queue upon the corresponding request message is served. This process uses the arrival time slot tracking process to identify redundant retransmissions. Finite buffer

case and lossy network conditions create non-redundant retransmissions which are joined the secondary (low priority) queue. Therefore, both arrival time slot tracking and removal processes are also considered for the secondary queue. The performance metrics namely the *failed call attempt* representing the ratio of the number of INVITE requests that cannot receive 100-Trying response within the request timeout timer to the number of the total call attempts and the *response delay* representing the duration between the transmission of original INVITE requests and the reception of the corresponding 100-Trying responses are derived using the equations of the proposed fluid-flow model. Because the calculation of the response delay uses arrival time slot tracking, this process is derived also for the conventional SIP. The formulation of infinite buffer case under lossy network conditions is derived for both the conventional SIP server and the PRSM. The experiments demonstrating the accuracy of models are presented in Chapter 6. We verify the accuracy of the models with the analysis and simulation tool. We also implemented the conventional SIP fluid-flow models for all modeling cases as a benchmark to quantify the performance improvements of the PRSM. The following experiments are conducted for each modeling case:

(a) overload caused by bulk arrivals,

(b) overload caused by server slowdown,

(c) overload caused by bursty arrivals.

For all test cases, we verified the accuracy of models that the proposed fluid-flow models can accurately capture the dynamic behavior of the PRSM. The results demonstrated that the PRSM performs better than the conventional SIP for all test cases.

3. To systematically study the SIP and PRSM behavior and verify the accuracy of the proposed fluid-flow models, we designed and implemented a discrete-event based analysis and simulation tool including SIP retransmission mechanism (with on/off capability), FIFO (First-In-First-Out) and strict priority queuing mechanisms in Chapter 5. Markov-Modulated Poisson Process (MMPP) is considered for both traffic modeling and service times to simulate bursty traffic arrivals and server slowdown behavior, respectively. The tool also supports emulating a lossy network

condition. The software structure is designed to be flexible for new extensions on queuing and traffic models. Its accuracy is tested according to theoretical expectations of MMPP for traffic and service times modeling. In addition to the performance evaluation of the PRSM, we used it to analyze the effect of SIP retransmission mechanism on a SIP server, including:

(a) The mean queue sizes when the first retransmission condition is met with respect to various buffer length values including pseudo-infinite. In this set of tests, the SIP retransmission mechanism is off but only the timer expiry for the first retransmission is recorded to check whether theoretical expectations are verified. The experiments are conducted for various traffic loadings such as various bulk arrivals and various service rates alternating between normal processing and maintenance modes (which implies a reduction in the processing capacity of the server), to create high retransmission probability.

(b) The relationship between blocking and retransmission probabilities is evaluated with respect to various buffer length values including pseudo-infinite. The tests are conducted with traffic loadings creating significantly higher number of retransmissions for both cases of bulk arrival and reduction in the server processing capacity.

(c) The effect of retransmissions on the arrival rate is evaluated by applying retransmissions to the server. The tests are conducted with traffic loadings creating significantly higher number of retransmissions for both cases of bulk arrival and reduction in the server processing capacity.

(d) The effect of retransmissions on the queuing dynamics such as mean queue length, blocking probability, and retransmission probability is evaluated. The tests are conducted with traffic loadings creating significantly higher number of retransmissions for both cases of bulk arrival and reduction in the server processing capacity.

(e) The ratio of the number of retransmissions to the number of the original SIP requests is evaluated with respect to various buffer length values including pseudo-infinite. The ratios are reported for each retransmission sequence such that the first retransmissions, the second retransmissions, etc., including the ratio of ceased retransmissions to the original requests which represents

the ratio of unsuccessful SIP calls to the original SIP session establishment requests. The tests are conducted with traffic loadings creating significantly high retransmissions for both cases of bulk arrival and reduction in the server processing capacity.

4. We designed and implemented a proof-of-concept in Chapter 7 using the open source JAIN-SIP stack to demonstrate that the implementation of the PRSM in real systems is feasible and requires minimal modifications. The implementation considered both server and client side requirements for individual tests, and a test under the SIP traffic created by SIPp traffic generator, which is also modified to include relevant retransmission information in request-line of SIP request messages. We observed that the original SIP without using PRSM crashed for all trials, while the PRSM completed all the tests without crashing. Statistical reports are collected from SIPp including number of successful calls, call failures and number of retransmissions. By using the same software for the conventional SIP, we also verified that the implementation of the PRSM can coexists with the conventional SIP design for the backward compatibility.

1.2 Organization of the Thesis

Following this introductory chapter, Chapter 2 provides an overview of SIP signaling concepts that are related to the content of the thesis, including SIP session establishment call flow, message and protocol structure. The proposed priority-based SIP request scheduling mechanism is presented in Chapter 3. Chapter 4 consists of discrete-time fluid-flow models for all cases; infinite buffer case in Section 4.6, finite buffer case in Section 4.7, lossy network case in Section 4.8, following the definition of common aspects such as the arrival time slot process in Section 4.4 and the removal process in Section 4.5. The retransmission analysis and simulation tool is presented in Chapter 5. After providing a brief background for MMPP in Section 5.1, the functional and software architecture are given in Sections 5.2 and 5.3, respectively. The accuracy analysis of the tool is performed in Section 5.4. Chapter 6 present the experiments conducted for fluid-flow models, comparing the conventional SIP and the PRSM. Chapter 7 consists of the detailed design and implementation of the PRSM in JAIN-SIP stack, as well as the performance evaluation results under the real-time

traffic generated by the SIPp traffic tool. Chapter 8 concludes the thesis and includes suggestions for the future work.

2. SIP BACKGROUND

SIP is an application layer protocol for establishing, modifying and managing sessions over IP networks. It is used in conjunction with other application layer protocols such as Session Description Protocol (SDP) [10] for session negotiation, Real Time Protocol (RTP) [11] and Secure Real Time Protocol (SRTP) [12] for media transmission, and Transport Layer Security (TLS) [13] for securing SIP messages. SIP has been standardized by Internet Engineering Task Force (IETF) with RFC 3261 [1] and adopted by 3GPP, ITU-T and ETSI as the core signaling protocol for IP Multimedia Subsystem (IMS) in next generation networks.

Currently, three type of service providers deployed SIP in their networks, traditional telecommunications providers, over-the-top alternative providers, and enterprises [14]. Telecommunications providers typically do not deploy SIP by providing soft-clients to the end users, but rather convert to and from Public Switched Telephone Network (PSTN)-based technologies by using telephony gateways and multimedia adapters at the edge. From a SIP signaling perspective, most of them supports only basic core SIP feature. Over-the-top providers, generally, provide end-to-end SIP applications by supporting soft-clients on computers or smart-phones with less focus on voice and more on other forms, such as presence, instant messaging, voice mail, etc. Enterprises generally uses SIP through Private Branch Exchange (PBX)-type systems to construct a converged network carrying voice and data in a single network. Most enterprises support basic properties of SIP as similar to telecommunications providers. Global enterprises also use SIP on other forms such as video conferencing, instant messaging, presence, etc., in their networks.

There are two basic SIP entities, SIP User Agents (UAs), and SIP servers. Two types of SIP servers are proxy servers for session routing and registration servers for UA location management. In a typical SIP session establishment scenario, User Agents (UAs) and SIP servers are two main SIP entities and the session between the caller (originating UA) and the callee (terminating UA) is set up through the support of

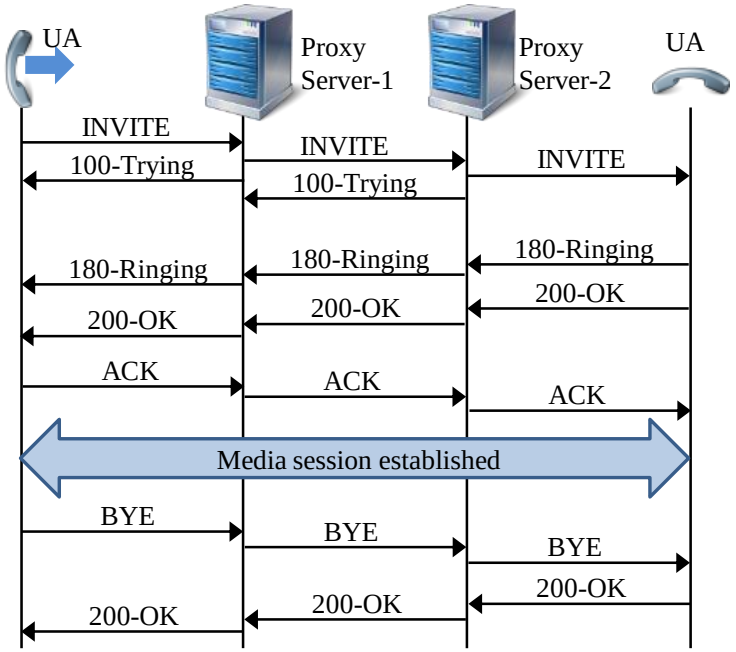


Figure 2.1 : The message flow for SIP voice call session.

SIP servers along the path between two UAs. UAs have externally visible identifiers, named SIP URI (Uniform Resource Identifier), for name-based routing which provides user mobility through the registration service. As shown in Figure 2.1, for establishing a voice call session, the caller sends an INVITE request towards the SIP proxy server (Proxy Server-1). The proxy server replies with a 100-Trying message and forwards the request to the Proxy Server-2 using a name-based application level routing, where the terminating UA can be reached. Proxy Server-2 sends 100-Trying response to inform the upstream SIP entity that the message has been received. When the terminating UA (callee) receives the INVITE request, it replies with a 180-Ringing message indicating receipt of the call request. This indication may be used to apply a ring-back tone at caller's UA device. When the callee picks up the phone, it sends a 200-OK message. The 200-OK message needs an ACK from the originating UA to complete the SIP three-way handshake that establishes a SIP session also known as a dialog. During this message exchange, SIP messages carry SDP information in the message body to provide an offer/answer exchange of media capabilities. Subsequently, media may flow between the originating UA and terminating UA without the intervention of SIP servers. Media stream is carried through RTP (or SRTP) which provides a container for media packets in real-time sessions for transport over IP networks. RTP uses a bit oriented header for efficiency in transport and

provides a container for media including voice and video. Then, the call session can be disconnected by sending BYE request message, which will be replied with 200-OK response message for confirmation.

2.1 SIP Protocol Structure

SIP protocol structure is presented with four layers, such that each layer has fairly independent processing stages with only a loose coupling between them. From bottom to top, they are:

1. Syntax and Encoding,
2. Transport Layer,
3. Transaction Layer,
4. Transaction User (TU)

Syntax and Encoding defines the syntax and encoding of SIP messages. It is specified using an Augmented Backus-Naur Form (ABNF) grammar [15]. The Transport Layer is responsible for the transmission of messages over the network transports, which includes determination of the connection to use for a request or response in the case of connection-oriented transports such as TCP and SCTP, or TLS (Transport Layer Security). Because SIP is an application layer protocol, it can use any network transport protocol, but RFC 3261 mandates the implementation of TCP and UDP for all SIP elements.

The Transaction Layer handles application-layer retransmissions, matching of responses to requests, and application-layer timeouts when setting up and tearing down a session. A request and all corresponding responses define a transaction, which is a fundamental component of SIP signaling.

The Transaction User (TU) creates and cancels SIP transactions, and utilizes services provided by the transaction layer. It builds request messages to be sent if it runs in the client mode and receives requests and provide appropriate responses if it is in the server mode. The TU is defined also the "core" of a SIP entity which designates the functions specific to a particular type of entity such as UA core, proxy core.

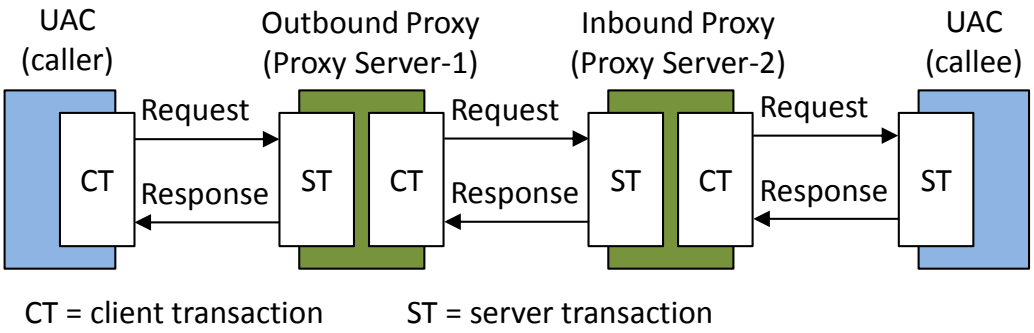


Figure 2.2 : An example of client and server transactions through SIP elements (based on Figure 4 in RFC 3261).

2.2 SIP Transactions

SIP is based on a request/response transaction model. Each transaction consists of a client request that invokes a particular function on the server side and at least one response. For example, in the previously given session establishment flow (Figure 2.1), INVITE request specifies the function that the caller wants to establish a session with the callee. More specifically, a SIP transaction occurs between a client and a server and consists of all messages from the first request message sent from the client to the server up to a final (non-1xx) response message sent from the server to the client. If the request is an INVITE message and the final response is a non-2xx message, the transaction also includes an ACK message to the response. The ACK message for a 2xx response message to an INVITE request message is a separate transaction.

1xx response messages are called as provisional responses, which is used by the server to indicate progress but that does not terminate a SIP transaction. Other responses are considered final. A SIP transaction may comprise zero or more provisional response messages.

A transaction created by the client to send a request is a client transaction, while a transaction along with all responses to that request is a server transaction. The purpose of the client transaction is to receive a request from the SIP element in which the client is embedded, and reliably deliver the request to a server transaction. The purpose of the server transaction is to receive requests and provide appropriate response(s).

There are two types of client transactions, depending on the method of the request. One that handles INVITE requests is an INVITE client transaction, and another type, which handles all requests except INVITE and ACK, is non-INVITE client transaction.

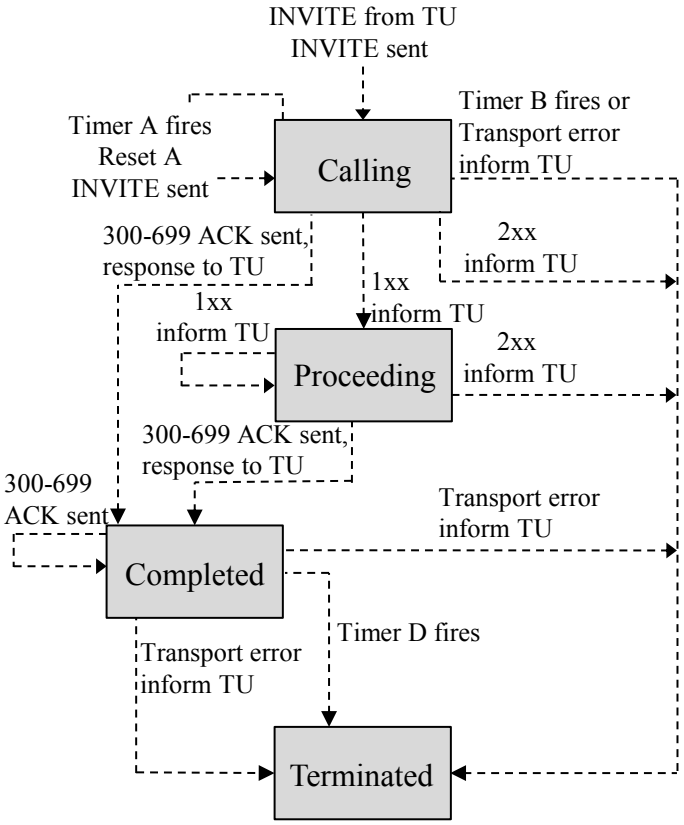


Figure 2.3 : INVITE client transaction finite state machine (RFC 3261).

As with the client transactions, server transactions are distinguished as INVITE and non-INVITE server transactions. Proxy servers, which are forwarding requests to the upstream servers or the terminating client, consists of both the client and server transactions for a session establishment. Figure 2.2 illustrates an example of the client and server transactions through the SIP elements of a session establishment.

RFC 3261 defines a finite state machine (FSM) for each of the four transactions. Figure 2.3 and Figure 2.4 depict the INVITE client and server transaction finite state machines, respectively. When Transaction User (TU) starts a session, the client transaction sends an INVITE message and enters the *"Calling"* state. In the case of using an unreliable transport, like UDP, it starts Timer A with a value of T_1 to control request retransmissions (refer to Section 2.4 for details). At the same time, for any type of transport, it starts Timer B with a value of $64 \times T_1$ to control the transaction timeout. In *"Calling"* state, if the client transaction receives a response indicating a progress (1xx, provisional response), it enters the *"Proceeding"* state and stops the retransmitting the request. In the *"Proceeding"*, any further provisional response is passed up to the TU without a change in the state. A response with status code from

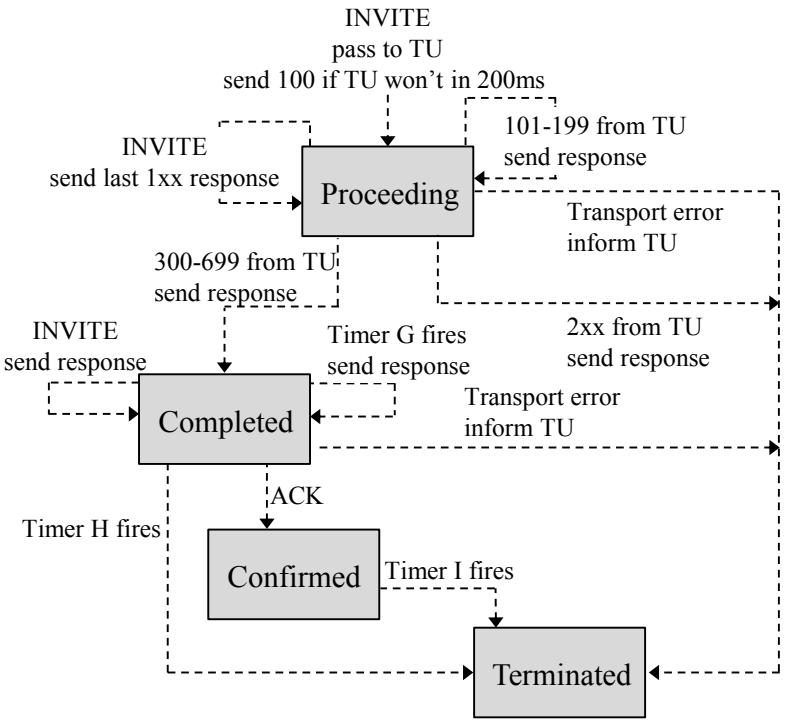


Figure 2.4 : INVITE server transaction finite state machine (RFC 3261).

300-699, in either *"Calling"* or *"Proceeding"* state, causes to enter the *"Completed"* state by sending an ACK message. Then, for unreliable transport, it starts Timer D with values of 32 seconds to handle retransmissions of 300-699 responses. A response with status code 2xx (final response), in either *"Calling"* or *"Proceeding"* state, causes to enter the *"Terminated"* state that the client transaction is destroyed. Sending ACK and receiving further 2xx response retransmissions, for unreliable transports, are handled by the signaling core without using a transaction.

When an INVITE request message is received by the server, if does not match to any existing transaction, the signaling core creates a server transaction with *"Proceeding"* state. The server transaction send a 100-Trying response unless it knows that the TU will generate a provisional or final response within 200 ms. This is necessary to quench request retransmissions rapidly in order to avoid network congestion. TU may still want to send a provisional response. In the case of a retransmitted INVITE request at *"Proceeding"* state, it re-sent the last sent response to the client. If TU wants to send a 2xx response, it enters to *"Terminated"* state to be destroyed. Retransmissions of 2xx responses are handled by the TU. States *"Competed"* and *"Confirmed"* are to complete the further message flow, if one of 300-699 responses sent by the TU.

2.3 SIP Message Structure

SIP uses text-based signaling messages using UTF-8 (USC (Universal Coded Character Set) Transformation Format-8) character set. The formatting of SIP messages is based on the syntax of Hypertext Transfer Protocol (HTTP) version 1.1. SIP message format consists of a start-line, multiple header fields, an empty line (a carriage-return line-feed sequence (CRLF) to indicate the end of the header fields, and a message-body, as follows:

```
SIP-message  =  <start-line>
               *<message-header>
                <CRLF>
                [message-body]

start-line   =  Request-Line / Status-Line
```

Request and response messages are distinguished by having a Request-Line or Status-Line in the start-line. The Request-Line consists of a method name (such as INVITE, BYE, etc.), a Request-URI, and the protocol version separated by a single space (SP) character:

```
Request-Line  =  Method SP Request-URI SP SIP-Version CRLF

Example:
INVITE <sip:abcd@example1.com> SIP/2.0
```

Method name determines the function that a request message wants to invoke on a server, such as INVITE, BYE, CANCEL. Request-URI identifies the destination of the request. SIP-Version indicates the version of SIP in use for the request message and it is set to "`SIP/2.0`" for RFC 3261.

Message headers provide additional information to the request message, such as subscriber identification, message routing information, transaction identification, branch identification, dialog identification, and contact address. The format of headers consist of a header-name and header-value which optionally may be multiple as separated with comma. The header value may consists of a semi-colon separated list of parameters.

```
header = "header-name" ":" header-value *("," header-value)

Examples:
Route: <sip:abcd@example1.com>, <sip:defg@example2.com>
Contact: <sip:abcd@example1.com>;expires=3600
```

In general, message-body consists of an application information to be exchanged between end-points and it is not used for signaling based information such as branch identification, transaction handling, etc. When SIP messages is used to exchange information about the media session to be established, the message-body is the SDP.

A SIP response message is constructed similarly to a request message, but instead of the Request-Line, it contains the Status-Line as the first line of the message. The Status-Line provides the result of the invocation of the request. The components of the Status-Line are the Status-Code and the Reason-Phrase as well as the SIP-Version. The Status-Code is a numerical representation (3-digit integer) of the outcome of the invocation of the request. Status-Code values represented 1xx are provisional codes indicating that the request received and continuing to process the request. 2xx are to indicate that the action requested by the message is completed successfully. A Status-Code if the form of 3xx indicates that the further action is needed to complete the request. 4xx, 5xx and 6xx are to indicate error situations. The Reason-Phrase is a short textual description of the Status-Code. It expresses the status code in a readable format. The Status-Line has the following format:

```
Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF

Example:
SIP/2.0 183 Session Progress
```

The following is a typical INVITE request message including an SDP message in the message-body:

```
INVITE sip:13@10.10.1.13 SIP/2.0
Via:  SIP/2.0/UDP 10.10.1.99:5060;branch=z9hG4bK343bf628;rport
From:  "Test 15" <sip:15@10.10.1.99>tag=as58f4201b
To:  <sip:13@10.10.1.13>
Contact:  <sip:15@10.10.1.99>
Call-ID: 326371826c80e17e6cf6c29861eb2933@10.10.1.99
CSeq:  102 INVITE
User-Agent:  Opensource Client
Max-Forwards:  70
Date:  Wed, 06 Dec 2009 14:12:45 GMT
Allow:  INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY
Supported:  replaces
Content-Type:  application/sdp
Content-Length:  258

v=0
o=root 1821 1821 IN IP4 10.10.1.99
s=session
c=IN IP4 10.10.1.99
t=0 0
m=audio 11424 RTP/AVP 0 8 101
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=silenceSupp:off - - - -
a=ptime:20
a=sendrecv
```

Headers `Via`, `To`, `From`, `Call-ID`, `CSeq`, `Max-Forwards`, `Contact`, `Content-Type` and `Content-Length` obtain the minimum required set that a SIP message shall include. `Via` header contains the address at which the caller is expecting to receive responses to this request. It also contains a branch parameter for identifying this transaction. `To` header contains a SIP URI to direct the request. It may also include a "display name" as a human readable name of the called agent. `From` header also contains a SIP URI to indicate originator of the request. The display name of the calling agent may also be included (`"Test 15"` in the example). The "tag" parameter of `From` header is used to identify the dialog which is started with the request. `Call-ID` contains a globally unique identifier for this call. `CSeq` (Command Sequence) header contains an integer (CSeq number) and the method name of the message. CSeq number is incremented for each new request within a dialog as indicating the sequence number of that new request. `Contact` header contains a SIP URI that represents a direct route to the originator. While the `Via` header field tells other elements where to send the response, the `Contact` header field tells other elements where to send future requests. `Max-Forwards` serves to limit the

number of hops a request can make on the way to its destination. It consists of an integer that is decremented by one at each hop. `Content-Length` includes an octet (byte) count of the message body. If the message contains a messages body, such as `Content-Length` indicates a number other than zero, the message shall contains also a `Content-Type` header indicating the type of the message-body. Any message header may appears any position in a SIP message, there is no a specific request for relative ordering for headers [1].

The functionality of headers in both the response and request messages are identical. However, there may be different headers present in a response message that are specific to it. Headers, such as `Via`, `CSeq`, `Call-ID`, that are used for SIP message routing and transaction handling, are copied from the request message. The server sending the response message is needed to add a tag parameter to `To` header to be incorporated by both endpoints into the dialog and will be included in all future requests and responses in this call. In general, the combination of the `To` tag, `From` tag, and Call-ID completely defines a peer-to-peer SIP relationship. The message-body of a response message also has the same functionality as the message-body of a request message. The following is a typical 100-Trying response message sent to the INVITE request given above:

```
SIP/2.0 100 Trying
Via:  SIP/2.0/UDP 13@10.10.1.13:5060;branch=z9hG4bK34456777
From:  "Test 15" <sip:15@10.10.1.99>tag=as58f4201b
To:  <sip:13@10.10.1.13>tag=77563384
Call-ID: 326371826c80e17e6cf6c29861eb2933@10.10.1.99
CSeq:  102 INVITE
Contact:  <sip:13@10.10.1.13; transport=udp>
Content-Length:  0
```

2.4 SIP Retransmission Mechanism

To guarantee reliable delivery of signaling messages, SIP retransmits lost or delayed signaling messages either end-to-end or hop-by-hop [1]. SIP utilizes two types of retransmission procedures, one for INVITE transaction, and the other for non-INVITE transactions (such as BYE - 200-OK transaction). For INVITE client transactions,

[1] RFC 3261 still recommends that headers which are needed for proxy processing (Via, Route, Record-Route, Proxy-Require, Max-Forwards, and Proxy-Authorization, for example) appear towards the top of the message to facilitate rapid parsing.

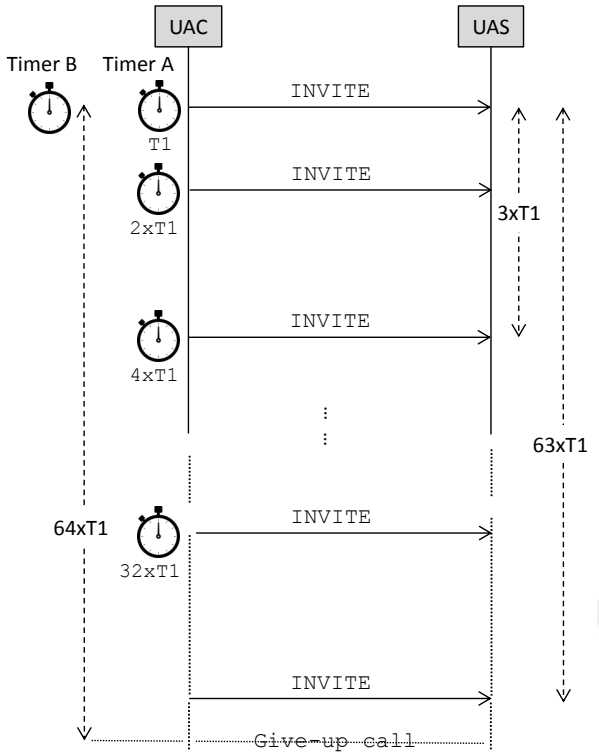


Figure 2.5 : INVITE transaction retransmission message flow.

RFC 3261 defines Timer-A, which controls the request retransmissions, with timer value T_1 for the first retransmission. If an unreliable transport is being used, as shown in Figure 2.5, the client transaction starts Timer-A with a value of T_1. When Timer-A expires, the client transaction retransmits the request by passing it to the transport layer, and resets the timer with a value of $2 \times T_1$. When Timer-A expires $2 \times T_1$ seconds later, the request is retransmitted again. This process continues so that the request is retransmitted with intervals that double after each transmission to create the exponential backoffs on retransmissions. During this processing phase, the INVITE client transaction is in the *"Calling"* state and it continues until receiving a 1xx response (the transaction state transitions to the *"Proceeding"* state) or expiry of Timer-B. Timer-B controls INVITE transaction timeouts and used not only for unreliable transports but for any transport. It is started when the original INVITE request is sent with a value of $64 \times T_1$. If the client transaction is still in the *"Calling"* state when Timer-B expires, the INVITE client transaction informs the TU that a timeout has occurred. The value of $64 \times T_1$ is equal to the amount of time required to send seven requests (one original request and 6 retransmissions) in the case of an unreliable transport.

T_1 is an estimate of the RTT (round-trip time) between the client and server transactions and the recommended default value is 500 ms. Nearly all of the transaction timers described for SIP signaling scale with T_1, and changing T_1 adjusts their values.

For unreliable transports, non-INVITE requests are retransmitted at an interval which starts at T_1 and doubles until it hits T_2. Then, retransmissions continue with T_2 intervals, even if a provisional response (1xx) is received, until a final response (2xx). The default value of T_2 is 4s. In this case, for non-INVITE transactions, there will be 11 retransmissions until the transaction timeout, $64 \times T_1$.

2.5 SIP Retransmission Detection with Transaction Matching

SIP is based on a request/response transaction model which is similar to HTTP [1]. A transaction comprises a request which is sent by a client to a server and all of the responses to that request are sent back from the server. According to RFC 3261, the transaction layer handles matching of responses to requests and application-layer retransmissions. Redundant request retransmissions are filtered and not passed into the upper layer. SIP transactions are identified by a combination of *"branch"* parameter and *"sent-by"* field in topmost Via header (which contains an IP address or host name, and port) and the request method (such as INVITE) as defined in Section 18.1.1 of RFC 3261. This identifier is used by the server to find transactions under processing and to decide if they are new requests or retransmissions. To achieve this task, parsing a SIP message is required, at least to find and parse topmost Via header. Transactions are governed by a finite state machine in Transaction Layer. For example, when an INVITE server transaction is created, it enters the *"Proceeding"* state. When it is in this state, TU may request any number of provisional responses from the server transaction to be sent through the tranport layer. These provisional responses do not cause a change in the state of the server transaction. If a request retransmission is received in the *"Proceeding"* state, the server transaction regenerates and retransmits the response as if it were replying to the first instance of the request.

If the upper layer sends a response back with status code from 300 to 699, the transaction enters *"Completed"* state and waits for an ACK [2]. Upon receiving an ACK

[2]However, if the sent response code is 2xx, the transaction immediately enters "Terminated" state. This behavior is a misidentifying situation that retransmissions of the request as a new, unassociated request and is corrected with RFC 6026 by modifying INVITE transaction state machine [16].

(or timer is expired), the transaction enters *"Terminated"* state. If a received request transaction matches with a transaction under processing, i.e. has a state other than *"Terminated"*, it is detected as a retransmission and Transaction Layer retransmits the most recent provisional response back.

SIP uses 100-Trying provisional response to prevent redundant retransmissions. When a transaction is constructed for an INVITE request, the server transaction must generate 100-Trying response to quench request retransmissions rapidly to avoid network congestion, unless it knows that TU will generate a response within 200 ms. As different from the other provisional responses, the 100-Trying response is never forwarded upstream by a stateful proxy. The retransmission detection rule is not applied to stateless User Agent Server (UAS). If a stateless UAS receives a retransmission of a request, it just forward to the downstream server. Furthermore, a stateless proxy does not generate its own 100-Trying response.

3. PRIORITY BASED SIP REQUEST SCHEDULING MECHANISM

The regenerative nature of the retransmission mechanism increases likelihood of SIP overloading [2]. When the message arrival rate including retransmissions exceeds the message processing capacity of a SIP server, the queue size increases and eventually the SIP server can be overloaded. This may result in a high queuing delay and trigger further redundant message retransmissions which may eventually cause the SIP server crash [3, 17–20]. A response message corresponding to a redundant retransmitted message is also redundant. Therefore, such redundant retransmissions increase the loads of both the overloaded server and its upstream servers. Thereby, retransmissions that caused by the overload rather than the message loss, bring extra overhead instead of reliability to the network and exacerbate the overload [7]. In this way, the overload situation may propagate from server to server, and eventually bring down the entire network [19].

Existing SIP can detect retransmissions as based on transaction matching algorithm [1]. The transaction layer handles application-layer retransmissions when the server transaction is in the *"Proceeding"* state. When a server transaction is constructed for a request, it enters the *"Proceeding"* state. The server transaction generates a 100-Trying response unless it knows that the Transaction User (TU) will generate a provisional or final response within 200 ms to quench request retransmissions rapidly in order to avoid network congestion. If a request retransmission is received while in the *"Proceeding"* state, the most recent provisional response that was received from the TU is passed to the transport layer for retransmission. Because this mechanism needs a server transaction for detection of corresponding retransmissions, it cannot detect redundant retransmissions before processing the message, in the case of excessively delayed messages due to overload in the server. This means that the server needs to parse a received message before deciding if it is a retransmitted request. Although SIP RFC recommends using the Via header field (and other header fields for proxy processing) appearing towards the top of the message to facilitate rapid parsing

(Section 7.3.1 of [1]), this ordering is not mandatory and in general SIP allows the relative order of header fields. Further, it is not a common practice in implementations. Therefore, parsing of a SIP request for retransmission detection, even if it is intended for finding the Via header, can be considered as parsing of a SIP request entirely.

In fact, SIP message parsing is a time consuming process. Although the text based message structure is attractive for readability and development, SIP has a complex context-sensitive grammar which allows various combinations for representing a valid SIP message [21]. It allows multiple legal representations of headers in a SIP message, i.e. short form header name and long form header name, some headers to be separated by commas or a carriage return line feed etc. The order of parameters within a header can vary. Both header and parameter names become SIP keywords that parsers must check depending on the message context [4]. Because of it is text-based, SIP entities need to perform string operations to allocate, copy, compare, append, and deallocate textual protocol information. As a result, lots of inefficient branches and memory copy are performed in SIP message parser [22]. Cortes et al. reported that 25% of the total processing time is devoted to parsing [4]. The same ratio is reported also by [5] for an open-source proxy, SER (SIP Express Router), parsing performance measurements, while [6] reported it as 20% for SER, 40% for JAIN-SIP based stateful proxies. Therefore, the existing retransmission detection mechanism is a time consuming process that affects the server's processing capacity in the case of redundant retransmissions.

In this thesis, motivated by the above aspects summarized up to this point, we propose a novel priority based SIP server request scheduling mechanism (PRSM) to increase the scalability of SIP servers by diminishing the negative effect of redundant retransmissions. The main idea is to identify retransmitted messages before queuing and to provide strict priority for the original incoming SIP requests over the retransmissions. For the message identification purpose, the server has to parse a minimal amount of the SIP message. To achieve this objective, the PRSM defines a parameter in the SIP request-line consists of the retransmission sequence number and the request identification. This parameter is used to identify retransmissions with a minimal message parsing effort. This identification is necessary to place an incoming request to its corresponding priority queue such that a new request to

the high priority (primary) queue and a retransmission request to the low priority (secondary) queue. When a request message is started to be served, the corresponding redundant retransmissions are removed from the low priority queue. Furthermore, this information can also be used to deploy various request scheduling mechanisms to improve the overall system performance.

The PRSM does not describe an overload control mechanism, implicitly, but it improves the performance of a SIP server by reducing the number and the effect of redundant retransmissions. It is expected that a priority based scheduling provides low delays for higher priority traffic, but if a large portion of the network traffic consists of high-priority traffic, the performance for the low priority traffic will be degraded. However, in our case, the redundant retransmission traffic occurs only if the queuing delay at the SIP server is above the first retransmission timer. Numerical results illustrate that the PRSM limit the number of the first transmissions by preferring to serve the original incoming requests first. Under the lossy network conditions, where retransmissions may be non-redundant, delay can be increased for call establishment attempts since non-redundant retransmissions due to lost requests are served in low priority. Investigations of other scheduling algorithms to reduce delays of non-redundant retransmissions will be considered as future works.

In the following, after summarizing the related work in Section 3.1, we describe the PRSM in Section 3.2 as based on high-level blocks of a typical SIP implementation.

3.1 Related Work

As a result of the works addressing the SIP overload control using 503-Service Unavailable response, RFC 5390 has been released to present problem statement and requirements for a possible solution. Correspondingly, IETF SIP Overload Control (SOC) working group is established in 2010. RFC 6357 [3] reports design considerations for a SIP overload control mechanism. Specifically, the local and the distributed SIP overload control mechanisms are discussed. The local overload control is an internal mechanism which is used to keep the performance of the server in an acceptable level under overload conditions. The mechanism monitors the resources of the server and a local overload is reported when the queue length [19, 23–26] or CPU load is grater than the pre-specified thresholds [19,27]. The queuing delay is also

used as an indicator for overload condition [8, 9, 20, 24]. When the local overload is detected, the conventional approach rejects request messages with 503-(Service Unavailable) response messages or just drops them without sending any response. Since retransmissions cannot be stopped in this way, this approach is not capable of preventing a congestion collapse in the server [3, 20]. Furthermore, rejecting the requests may cause a network wide congestion collapse [2]. Therefore, local overload control cannot be efficiently used for SIP overload control but may be used in conjunction with distributed overload control mechanisms which consider overload control between network elements [3].

The second approach, distributed overload control mechanism, which is named also as the external method, can be explicit or implicit. The former sends a signal to the upstream network elements to indicate that the server is reaching its capacity limit for adjusting the transmissions rate according to the information in the received signal. RFC 6357 defines fundamental components of a system using the explicit overload control mechanism. The possible delay on the feedback signal reaching to the upstream server is considered as a drawback of the approach [28]. The later uses metrics like transaction delay, session setup delay as implicit feedback. There is no need for modification in SIP signaling that can be applied to the communication with non-SIP network elements.

The distributed overload control mechanism can be used in hop-by-hop or end-to-end modes. In hop-by-hop mode of operation, the overload situation feedback is sent to the next hop only in the network. End-to-end mode of operation defines an overload control loop along the entire path of a SIP request. It consolidates overload information from all SIP servers in the path and uses this information to throttle traffic if needed. An overload control mechanism working on end-to-end mode has to be able to frequently collect the overload status of all servers on the potential path(s) to a destination and use this data to obtain an useful overload feedback.

IETF SOC working group concluded their work in 2015 by considering two complementary approaches for handling overload situations for server-to-server topologies: the first one with flow sifting on the sender side (loss-based overload control, [29]) and the second one with restricting the flow rate of signaling messages (rate-based overload control, [30]).

The former, RFC 7339, defines SIP signaling parameters in Via header and the behavior of SIP servers for overload control while specifies a loss-based overload scheme as the default algorithm to be supported by servers comply with overload control. The *"oc"* parameter is suggested as an indication in the topmost Via header from the upstream server to the downstream servers that it supports overload control. The parameters also indicate the level of control to be applied if the overload condition occurs. The *"oc-algo"* parameter is defined to include supported overload control algorithms (such as "loss-based", "rate-based") by the upstream server. The *"oc-validity"* parameter is inserted by the downstream server in a response to indicate the duration that the load reduction specified in the value of the *"oc"* parameter to be considered. The *"oc-seq"* defined as the parameter to indicate the sequence number used to differentiate two *"oc"* parameter values generated by an overload control algorithm at two different instants in time. For example, consider an upstream server, ProxyA sends an INVITE request including overload control parameters to a downstream server ProxyB, and receives the corresponding 100-Trying response, as follows:

```
INVITE sip:user13@upstream1.com SIP/2.0
Via: SIP/2.0/UDP ProxyA.upstream1.net;branch=z9hG4bK343bf628;rport;
oc;oc-algo="loss,alg2"
............

SIP/2.0 100 Trying
Via: SIP/2.0/UDP ProxyA.upstream1.net;branch=z9hG4bK343bf628;
received=10.11.12.133;oc=15;oc-algo="loss";oc-validity=500;
oc-seq=112
............
```

In the signaling flow above, since ProxyA supports overload control, it inserts the *"oc"* parameter in the topmost Via header of INVITE request. ProxyA supports two overload control algorithms called *"loss"* and *"alg2"*. In the 100-Trying response message, because ProxyB also supports overload control and prefers loss-based algorithm it sends *"loss"* back to ProxyA in the *"oc-algo"* parameter. Since it is overloaded, it sets *"oc"*, *"oc-validity"* and *"oc-seq"* parameters to 15, 500 and 112, respectively. The value 15 in *"oc"* parameter indicates that ProxyA should decrease the messages sent to ProxyB by 15% for 500 milliseconds. The sequence number shall be increased according to the previous overload indication. In the case of there is no

overload in ProxyB, it sets *"oc"* and *"oc-validity"* parameters to 0 and omits *"oc-seq"* parameter, while *"oc-algo"* still indicates loss-based algorithm.

The later, RFC 7415, proposes a rate-based control scheme by using the same signaling suggested in the former, RFC 7339. By arguing that a loss-based algorithm is sensitive to variations in load so that any increase in the offered load would be directly reflected by upstream servers to the overloaded servers, it suggests to communicate the target maximum SIP request rate as in messages per second. In this case the *"oc"* parameter of responses includes the *"rate"* information to indicate the suggested rate. It proposes a default algorithm to throttle new SIP requests at the rate specified by "oc" parameter in conformance with the upper bound of $1/T$ messages per second, where $T = 1/[\text{"}oc\text{"}parameter]$. The default algorithm is based on the leaky bucket algorithm in [31].

Another RFC published by SOC working group, RFC 7200 [32] defines a "load-control" event package to distribute load-filtering policies to other SIP entities in the network. With this event package, a sending (upstream) entity can subscribe to the overload status of its downstream neighbors and receive notifications of overload control status changes with SIP NOTIFY request messages.

As stated in RFC 6357, since the overload control is based on reducing the number of messages a sender is allowed to send, it is only applicable for server-to-server topologies but not suited for client-to-server topologies, named also as "edge proxy" topology in RFC 6357 (refer to Figure 3.1 for discussed topologies in RFC 6357), such that servers which receive requests from a very large population of UAs, each of which only send a very small number of requests. Although UAs typically initiate a single request at a time, a large number of UAs can cause overload on edge proxies for some situations such that "avalanche restart", "flash crowds" scenarios described in RFC 5390.

Recent studies on SIP overload control, in general, are based on the given design considerations of SIP overload control. [33] presents the loss based overload control mechanism developed by IETF SOC working group, which uses hysteretic load control technique on the server side for preventing overloading. They follow an analytical modeling to construct and analyze SIP server model in form of queuing system

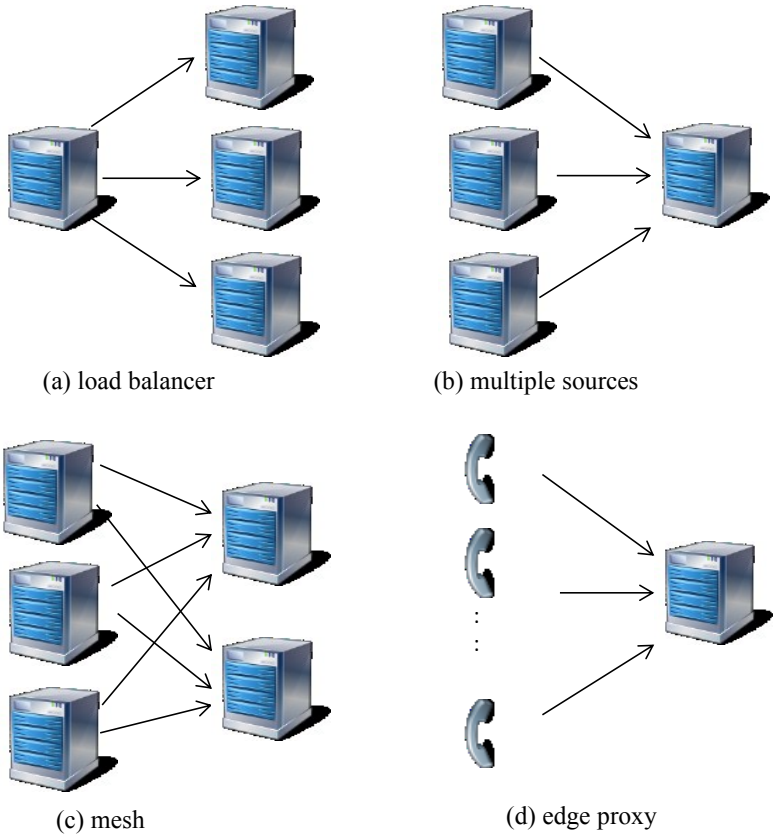


Figure 3.1 : SIP server topologies discussed in RFC 6357.

with finite buffer occupancy and two-level hysteretic overload control. In [28], a distributed end-to-end adaptive window based overload control algorithm, which does not receive an explicit feedback from the downstream server, is proposed. Since the upstream servers use call establishment delay as a measure of the amount of load on the downstream server, it is claimed that the proposed algorithm imposes no additional complexity or processing on the overloaded downstream server, that improves robustness.

A cooperative overload control method is developed in [34] to detect and control overload in SIP servers, as based on the system model specified in [3]. The algorithm decides how much traffic to be sent to upstream server based on the information received from neighboring servers, where the information is received within OPTIONS SIP method. [35] proposes a coupling of local and distributed overload control mechanisms which allows detection of sophisticated overload conditions such as those caused by non-SIP entities. The distributed overload control mechanism uses transaction response time as the basis for implicit overload detection. [36] studies the SIP system behavior by separating signaling traffic in two different classes (INVITE

and Non-INVITE) and creating a cut-off priority queuing model. In order to reduce the rate of retransmissions, when an upstream server receives a feedback about overloading from the downstream server, it removes requests from its queue exceeding a fixed timeout.

In the literature prioritization is also used as an approach to reduce the effect of overload conditions. SIP messages are prioritized, in general, according to their role in SIP session establishment, such as higher priority is given to INVITE messages over non-INVITE messages and vice versa [23, 25, 36–38]. Another approach is removal of retransmissions rather than giving prioritization. Under this framework, [39] proposes a front-end SIP flow management system (FEFM) which integrates flow rate limiting, message scheduling, and admission control to achieve overload protection, service differentiation, and performance management. The presented technique performs response time prediction, multi-queue scheduling, and retransmission removal. The module Redundant Request Remover (R^3) of FEFM uses a recording mechanism to monitor requests delivered to the server but neither got reply nor expired. When a new request is received, the server first checks if it is already in the recorder. If so, the request is deemed as a retransmission and dropped immediately. In order to improve server performance with the help of offloading SIP message parsing, SIP transaction processing, security and transport layer connection management, in [40], a SIP Offload Engine (SOE) capable of retransmission removal flexibility is presented. In [41] both removal and prioritization are suggested. This is achieved with a priority queuing model to reduce the retransmissions by giving high priority to retransmitted requests and to avoid transmitting unnecessary retransmissions towards downstream of SIP network. However, the retransmission identification approach (based on call-Id) used in [41] does not comply with the SIP standards, since call-Id does not identify a transaction but a session, such that, some other INVITE messages can be sent by the sender for an existing session to modify session parameters, dialog parameters, or both [1, 42]. All these works need parsing the entire message that requires significant amounts of processing time as it is evaluated in [4, 5, 39], that they may not differ from the behavior recommended for the conventional SIP at Section 8.2.7 of [1] except discarding the message instead of resending the last response, so an

approach eliminating the parsing effort may bring additional improvement to the server performance.

Our approach is different than these studies since it does not require a feedback signaling between servers to prevent the server crashing due to overloads, by proposing a redundant retransmission identification mechanism to decrease redundant processing power in overloaded SIP servers. Next subsection describes the PRSM.

3.2 The Proposed Priority Scheduling Mechanism

The idea behind the PRSM is that if the amount of the queuing delay at the SIP server is above the first retransmission timer, it is preferred to serve the original incoming request first to limit the number of the first retransmissions and hence the number of the subsequent retransmissions in the network. This is achieved by including additional modules into the existing SIP message processing scheme. Figure 3.2a shows the standard modules in the receive path of the SIP to process raw SIP messages received from the UDP socket of the network interface. The incoming messages are placed into a FIFO queue irrespective of whether they are new requests or retransmissions. The SIP server parsed them according to the FIFO queuing discipline and forwards the successfully processed messages to Transaction Layer. As shown in the figure, new requests and retransmissions are treated equally since the FIFO queue is employed.

Figure 3.2b depicts the high level architecture of the PRSM which has three new functional modules: *Request Pre-parsing Module*, *Priority Queuing Module*, and *Retransmission Removal Module*. In the following subsections, these modules are described in detail.

3.2.1 Request pre-parsing module

The *Request Pre-parsing Module* decides whether an incoming SIP request is a new arrival or retransmission with a minimal parsing effort. This classification is necessary to place an incoming request to its corresponding priority queue (e.g., a new request to the high priority queue and a retransmission request to the low priority queue). To the best of our knowledge, there is no functionally equivalent module of *Request Pre-parsing* in the existing SIP standards or literature. In order to reduce message parsing effort, we propose an extension on SIP request-line which assumes a parameter

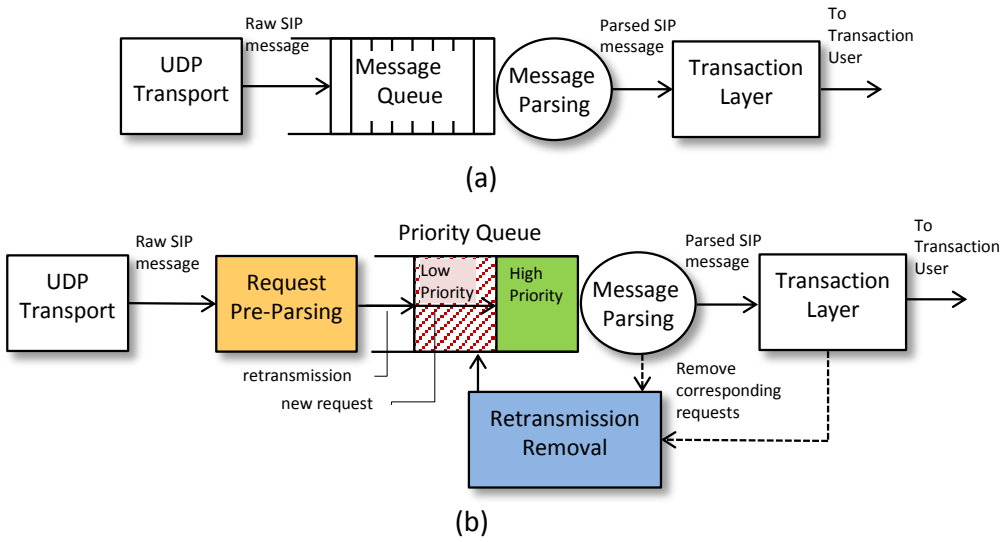


Figure 3.2 : (a) High level functional view of a typical SIP implementation up to Transaction Layer,
(b) Its extension with the PRSM.

indicating if the request is a retransmission, with an additional request identification parameter, similar to the branch parameter of the Via header. For this purpose, we defined a new parameter called "rb" (retransmission and branch) in SIP request-line. The syntax of "rb" parameter with example usage is as follows:

```
rb = "rb" EQUAL rseq-num ":" r-branch
rseq-num = 1*DIGIT ;retransmission sequence number
                   ;(0 represents the original request
r-branch = token   ;same as via-branch at [1]

Example:
INVITE sip:service@127.0.0.1:5070 rb=0:z9hG4bK-25466-1330-0 SIP/2.0
```

With this definition, the parameter "rseq-num" includes two types of information; if the request is a retransmission (i.e., "rseq-num" is not "0") and the sequence number (ordinal rank) of the retransmission. If the retransmission sequence number ("rseq-num") is zero, this SIP request corresponds to the original transmission. If "rseq-num" is one, it corresponds to the first retransmission, so on.

The transaction identifier ("r-branch") has the same format defined for the "branch" parameter of the top-most Via header and begins with "z9hG4bK" magic cookie as defined in RFC 3261, which indicates SIP compliance. "r-branch" is needed to identify redundant retransmissions when they will be removed by the *Retransmission Removal Module*. In RFC 3261, for uniqueness across all transactions,

the key for transaction matching is obtained by combination of "branch" parameter of topmost Via header, "sent-by" parameter of topmost Via header, and the method of the request (such as INVITE (except ACK)). Since the method of the request is already obtained from the request-line, "r-branch" parameter may be extended to include a sender based information (like "sent-by" value) to prevent accidental duplication from different senders. In this thesis, we consider "r-branch" is the same as the "branch" parameter of the top-most Via header, and no extension parameters included like "sent-by".

The cookie "z9hG4bK" of the "branch" parameter is defined to distinguish old SIP clients complying with old SIP RFC (RFC 2543) [43], which recommends to use other information, such as Call-Id, and suggest to support these clients for backward compatibility. In this thesis, we do not consider SIP clients complying with RFC 2543.

Since UDP is a message-oriented protocol, it provides a single complete message [44] to the Request Pre-Parsing module; this means that Request Pre-parsing does not require extra processing to find message boundaries. When a received message is identified as a request or response, and a new arrival or a retransmitted request, the message is inserted into the corresponding priority queue. However, for example, since TCP is a stream-oriented protocol, the receiver needs to parse the message when receiving from the network interface to find Content-Length header, which is commonly the last header of the message (refer to Section 2.3), for determining the message boundaries before queuing.

The request-line parser of *Request Pre-parsing Module* needs to consider coexistence of clients supporting both the conventional SIP and the PRSM. As it can be seen in the proof-of-concept implementation by using JAIN-SIP stack (Chapter 7), this is applicable without a change on SIP version number, such as the parser needs to count tokens of the request-line, which are separated by space. If the number of tokens is four, this means that the request-line has the "rb" parameter. In the absence of the "rb" parameter, the request message is considered as high-priority.

3.2.2 Priority queuing module

We consider that the *Priority Queuing Module* implements a non-preemptive strict priority queuing to minimize the response delay of new (original) requests. Messages which have the same priority are queued in a FIFO (First-In-First-Out) manner. Priority queues may share the same buffer or use separated buffers. Buffer size(s) can be set to a finite size or in pseudo-infinite where the buffer size limited by the availability of computing resources (note that, in practice, all queues are finite, then we use pseudo-infinite term to imply there is no buffer size setting by the configuration). In the case of a single buffer shared by the priority queues, for example, a new arrival request, which has the highest priority, enters the tail of the queue prior to retransmitted requests.

In fact, once we have the retransmission sequence number in the message, we may apply some other priority rules such as providing the highest priority to retransmissions or using a prioritization as based on the retransmission sequence number. We performed extensive experiments using our retransmission analysis and simulation tool and observed that for higher buffer sizes, providing the highest priority to the original messages yielded better performance results compared to other approaches and more decent to investigate the performance of the PRSM. Therefore, in this thesis, we consider only the prioritization that the original request messages have the highest priority. For sake of simplicity, we do not consider any precedence within the retransmit sequence. Messages having no retransmission information in the request-line are processed as the highest priority messages.

Note that, relatively small finite buffer sizes can be used as a simple overload control mechanism with a cost of temporary call rejection rate increasing the retransmission rate, which may result in increased redundant use of network resources. Eventually, in these cases, the rate of non-redundant retransmissions will be increased accordingly. On the other hand, by considering that priority scheduling provide low delays for the high-priority traffic, but if a large portion of the network traffic consists of high-priority traffic, the performance for the low-priority traffic can be severely degraded [45–47].

3.2.3 Retransmission removal module

In our scheme, if a request has already been processed and "100-Trying" response message is returned back to the originator, the corresponding retransmission messages are removed from the queue to minimize the redundant usage of the limited server resources. The Retransmission Removal Module performs deletion of the corresponding redundant retransmissions from the low priority queue when a message is under processing. It can be invoked from either Message Parsing or Transaction Layer by using the request identifier (`"r-branch"`). Note that this module may have detrimental impact on the "reliability" of SIP signaling since the redundancies are eliminated. However, it provides significant improvement in server's processing capability as given in numerical results in Section 5.5 and Chapter 6. In addition, it should be noted that, retransmissions due to previously sent requests never reached to the Priority Queuing Module due to network or server congestion, are non-redundant and will be processed by the server if they can enter into the queue (but in lower priority). In the case of Transaction Layer invokes Retransmission Removal, Transaction Layer may select to send 100-Trying for the removed retransmissions with a slight degradation of the server performance.

The retransmission removal functionality is configurable, where one can set it on and off. When it is set to off, the performance improvements of the PRSM due to the prioritization mechanism can be measured without the removal of any retransmission message from the secondary queue. Note that serving the original incoming request first limits the number of the first retransmissions and hence the number of the subsequent retransmissions in the network. When the retransmission removal functionality is set to on, the additional improvement of PRSM by the removal of redundant retransmission can be observed.

The performance of the removal task depends on the data structure used in implementation approach. Since the `"r-branch"` parameter is text based, the search job in the data structure always requires character comparing. We have observed the average value of removal as 18.1μs for our JAIN-SIP implementation of the PRSM,

which takes 1.07% of the time requested for parsing of a request message entirely (refer to Section 7.4 for details).

4. DISCRETE-TIME FLUID-FLOW MODELS

In this chapter, we present fluid-flow model to study the performance of the PRSM under various server models and network conditions. There are three reasons for SIP retransmission: *(a)* excessive processing delay in the queue, *(b)* blocking of the original messages and subsequent retransmissions, and *(c)* message loss in the network. We consider the following three modeling cases to cover these reasons:

1. The server with infinite buffer size: The SIP server equipped with the PRSM has infinite buffer size for message queuing. To investigate the effect of redundant retransmissions through the model, we assume that there is no message loss in the network. Because of the infinite buffer size, there is no queue blocking in the server. Therefore, all retransmissions are because of excessive queuing delay, so redundant, and there are no non-redundant retransmissions in the system.

2. The server with finite buffer size: In this case, if the buffer is full, incoming messages are blocked and dropped with no further processing, i.e. no response is sent if the message is a request. Therefore, retransmissions corresponding to requests that all previously blocked are non-redundant. Consequently, in this case, we model and investigate the handling of non-redundant retransmissions by the PRSM. The lossless network assumption is still valid for this model.

3. The server with infinite buffer size under lossy network conditions: In this case we will model and investigate the effect of non-redundant retransmissions because of losses in the network.

The case with finite buffer under lossy network conditions consists of two types of non-redundant retransmissions, namely retransmissions because of blocking and loss, such that this case is just an cascade usage of cases (3) and (2), respectively. This case will be considered as a future work.

In general, for all cases, we assume a SIP proxy server receiving traffic from multiple upstream servers and/or UACs such that it can be an edge proxy.

We derive formulation for performance metrics namely the failed call attempts and the response delay from sending INVITE request until receiving a 100-Trying response, for all cases.

After providing the motivation for the analytical modeling approach in Section 4.1, we present assumptions and basics definitions (preliminaries) common for all cases in Sections 4.2 and 4.3, respectively. Section 4.5 presents an high-level view of removal process, following the *arrival time slot tracking process*, which is used by both removal process and calculation of response delay performance metric, given in Section 4.4. Sections 4.6, 4.7 and 4.8 are for modeling cases 1, 2 and 3, respectively. In Section 4.9, we derive formulation of performance metrics for all cases.

4.1 Motivation for Fluid-Flow Modeling

For the modeling of the PRSM, an essential component is the modeling of SIP retransmission traffic. Modeling retransmission traffic is a challenging task since the retransmission mechanism generates duplicate messages, which have a complex correlation structure due to various timers [7]. As the retransmission mechanism amplifies the SIP overload and makes an overloaded SIP server unstable, existing models created for a stable SIP system cannot be effectively used to analyze the dynamic performance of an overloaded server. In the literature, there are two main studies modeling SIP retransmission traffic: *(i)* the theoretical model of retransmissions and delays on session establishment due to overloaded servers by Sisalem et al. [48], and *(ii)* the discrete-time fluid-flow model to evaluate the impact of SIP retransmission mechanism on SIP server overloading by Yang Hong et al. [8] [9]. The former model is capable of calculating the total number of request messages, including retransmissions, transmitted by the sender at any point in time. The model considers losses due to the capacity limitation of the overloaded server or lossy links causing retransmissions and excessive delays in session establishments. The system eventually reaches a steady state since SIP retransmissions cease after $64 \times T_1$ seconds. The server capacity and link loss rate are set to fixed parameters and hence the model does not consider retransmissions due to excessive queuing delays. The latter model considers excessive queuing delays as the main reason for retransmissions and defines fluid-flow based models to capture the dynamic behavior of SIP retransmission

mechanism of a single server with infinite buffer [8] and tandem servers with finite buffer [9].

Since the PRSM is dealing with regenerative nature of the SIP retransmission mechanism because of redundant retransmissions as result of excessive queuing delays in an overloaded SIP server, for modeling the PRSM, we have selected the discrete-time fluid-flow based model, which is considered by [8] and [9]. Besides, discrete-time fluid-flow model provides a better view than the approach in [48], which considers the SIP server as a black-box with a constant capacity to model the removal mechanism of the PRSM.

In comparison to a model for the conventional SIP, a model for the PRSM needs to consider a priority scheduling discipline. Therefore, we consider a discrete-time single-server queuing system with two priority queues, namely the primary and secondary queues for the original and retransmitted requests messages, respectively. In addition, the PRSM has the redundant retransmission removal property that needs to be considered by the model. Without some information about the manner in which particular request message currently is processed that arrived a specific time and how long it waited for processing, it is not possible to estimate corresponding redundant retransmissions to be removed. To handle this challenge, first, we develop the *arrival time slot tracking process*, attached to the queuing process, for keeping track processed request messages, then we develop the *removal process* to determine corresponding redundant retransmissions for removal by using the *arrival time slot tracking process*.

4.2 Assumptions

Unless otherwise specified, the main assumptions of all the models are as follows:

(i) Transmission and propagation delays are negligible [24], and queuing and processing delays are dominant for retransmissions.

(ii) The scheduling mechanism provides strict priority for the original incoming SIP requests over retransmissions, i.e. messages in the secondary queue (contains retransmissions) are processed only if the primary queue (contains original request messages) is empty.

(iii) The non-preemptive service is performed by the server, i.e. the ongoing service cannot be interrupted when a higher priority job arrives.

(iv) Within the same queue, the service discipline is FIFO (First-In-First-Out).

(v) The time for pre-processing a message, which is needed to identify retransmissions before queuing, is negligible compared to the entire processing time of the message.

(vi) The time required for removing a message from a queue is negligible compared to the entire processing of the message.

(vii) Within a same time slot, an original request message has a higher priority over a retransmitted request message to join the queue (*enqueuing priority*)

(viii) Hop-by-hop INVITE-"100-Trying" transactions are the main contributer on triggering retransmissions. Therefore, for analytical simplicity, only hop-by-hop INVITE-"100-Trying" transactions are considered.

Because of discrete-time fluid-flow approach, time is divided into discrete time slots. The errors due to discretization of time can be reduced by setting the interval of a time slot to an arbitrarily small value.

4.3 Basic Definitions

In this section, basic definitions are provided from the PRSM point of view. Definitions related to the priority queuing and removal processes are not applicable to the conventional SIP cases.

In the discrete-time fluid-flow modeling, time is divided into small fixed-length discrete time slots denoted by n. Therefore, for all modeling cases of the PRSM, we consider a discrete-time single-server queuing system with two priority queues, namely the primary and secondary queues for the original and retransmitted requests messages, respectively.

Dynamics of the queuing model are characterized by six processes; namely, the arrival process, the service process, the remaining service capacity process, the arrival time slot tracking process, the primary removal process and the secondary removal process.

In this section, we define the first three processes. The arrival time slot tracking process is presented in Section 4.4. The primary and secondary removal processes are given in Sections 4.5.1 and 4.5.2, respectively.

The *arrival process* is determined by an aggregation of original and retransmitted INVITE request message rates, denoted by $\lambda(n)$ and $r(n)$ for time slot n, respectively. Here, the rate corresponds to the number of messages for each time slot. $r(n)$ is the aggregate rate of retransmitted messages at time slot n such as 1^{st}-time retransmissions, 2^{nd}-time retransmissions up to 6^{th}-time retransmissions [1]. Thus, the total retransmitted messages $r(n)$ at time slot n is:

$$r(n) = \sum_{j=1}^{6} r_j(n) \tag{4.1}$$

where $r_j(n)$ refers the j^{th}–time retransmission for the original request messages arrived at time $n - T_j$. Here, T_j is calculated according to exponential backoff algorithm, i.e. $T_j = (2^j - 1) \times T_1$ and $1 \leq j \leq 6$. T_1 is the first retransmission timer defined in the SIP standard with default value 500 ms. Note that, when used in a discrete-time formulation, the value of T_j shall be considered in discrete manner, such as for 50 ms time slot duration, the value of T_1 is 10.

Under the assumption that transmission delay is negligible, when the server processes a request message at the current time slot n, it sends a 100-Trying response to the originator which causes stop the retransmit timer. Therefore, retransmitted messages from originators would be related to unprocessed messages rather than processed ones at the current time slot n.

An original request message always joins the primary queue while a retransmitted request message always joins the secondary queue. $\lambda(n)$ is a given model parameter, while $r(n)$ is determined by the queuing dynamics of the server. $\lambda(n)$ may have any arbitrary distribution.

In the case of queue blocking or loss in the network, only $\lambda^e(n)$ portion of $\lambda(n)$, which is called an effective original arrival rate, can join the queue or arrive the server, while $\lambda^d(n)$ is dropped because of blocking or $\lambda^l(n)$ is lost. Similarly, $r^e(n)$ denotes effective retransmission rate, while $r^d(n)$ denotes dropped because of blocking and $r^l(n)$ denotes lost retransmission rate, respectively. Note that, B denotes the buffer size of the server for the finite buffer case.

The *service process* is determined by the service time of an INVITE message denoted by $\mu(n)$. A service time of a message is measured starting from parsing a received INVITE message until a 100-Trying response sent. Messages in the primary queue have a strict priority over those in the secondary queue, so that messages in the secondary queue can be processed only when the primary queue is empty. Messages within the same queue are processed according to FIFO discipline. The processing in the secondary queue is performed by the *remaining service capacity process* that denoted as $y(n)$. While $\mu(n)$ is a parameter provided for the model, $y(n)$ is determined by the queuing dynamics. $\mu(n)$ may have any arbitrary distribution.

For the queuing models, $q(n)$ denotes the total queue size, while the primary and secondary queue sizes are denoted by $q_p(n)$ and $q_s(n)$, respectively. Hence, we have

$$q(n) = q_p(n) + q_s(n). \tag{4.2}$$

For the formulation of queuing dynamics, assume $\lambda(n)$, $\mu(n)$, and B are given, the objective of a proposed model is to calculate $\lambda^e(n)$, $\lambda^d(n)$, $\lambda^l(n)$, $r^e(n)$, $r^d(n)$, $r^l(n)$, $q_p(n)$, $q_s(n)$, $q(n)$, where applicable, and other queuing dynamics such as queuing delays of both primary and secondary queues.

4.4 Arrival Time Slot Tracking Process

We define the arrival time slot tracking to keep track of the time slot of request messages when they arrived into the system and are picked for service processing. In general, at the current time slot n, if the primary queue is not empty, the server processes arrivals joined the queue at time slot $n-k$, where $0 \leq k \leq n$. Note that the effective arrival rate is $\lambda^e(n-k)$ at time slot $n-k$. Therefore, the removal process needs to track recursively the number of original request messages processed at the current time slot n while arrived at *"arrival time slots"* $(n-k)$, where $0 \leq k \leq n$. Depending on the processing capacity $\mu(n)$, the server may process: *(i)* a part of arrivals within a single arrival time slot, *(ii)* all of arrivals within a single time slot, or *(iii)* arrivals within more than one arrival time slot $(n-k),(n-k+1),(n-k+2),\ldots$

Let $u(n)$ be a process which tracks arrival time slot(s) whose messages have been processed at time slot n, such that $u(n) = (n-k)$. For the case *(i)*, the removal process also needs to track inner position, $v(n)$, of that arrival time slot. We assume that the set

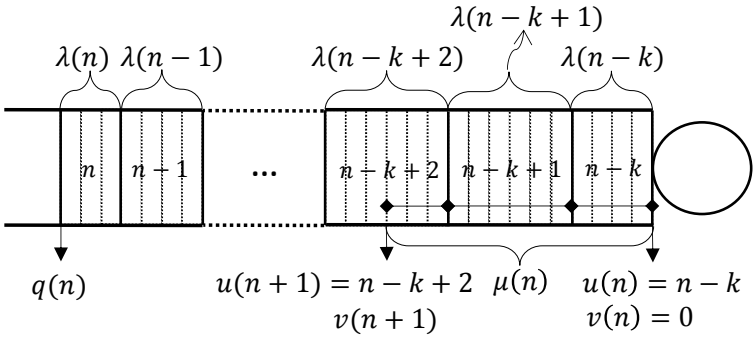


Figure 4.1 : An illustration of time slot tracking process for the primary queue.

S_n consists of how many arrivals are processed from which arrival time slots, such as $S_n = \{s_n(n-k), s_n(n-k+1), \dots\}$.

Figure 4.1 illustrates an example of the arrival time slot tracking for the primary queue. At the beginning of the time slot n, $u(n)$ indicates the arrival time slot $(n-k)$ to be processed, and $v(n)$ indicates the beginning of the arrival time slot $(n-k)$ (i.e. $v(n) = 0$). The processing capacity $\mu(n)$ has a value of 13, which is capable of processing the messages of the first two arrival time slots and the part of the third arrival time slot, from $\lambda^e(n-k)$ to $\lambda^e(n-k+2)$ with the values 4, 6 and 3, respectively. When the processing is completed, $u(n)$ indicates the arrival time slot $(n-k+2)$ for the next time slot $n+1$, and $v(n)$ indicates the third position of that arrival time slot. The process will generate a set for the time slot n, as $S_n = \{s_n(n-k) = 4,\ s_n(n-k+1) = 6,\ s_n(n-k+2) = 3\}$.

Generically, in service at time slot n, messages can be original request messages joined the queue at arrival time slot $n-k$, $\lambda^e(n-k)$, if it is the primary queue or retransmitted messages joined the queue at arrival time slot $n-k$, $r^e(n-k)$, if it is the secondary queue. In order to obtain a general expression, we use $L(n-k)$ to denote messages to be processed in the time slot n. The server processing capacity can be $\mu(n)$ or the remaining processing capacity $y(n)$ as depending on whether the queue is primary or secondary. Then we use $M(n)$ to denote the processing capacity at the time slot n.

In the arrival time slot tracking process, it is assumed that at current time slot n, the server is processing arrivals joined at the time slot $u(n) = (n-k)$ and the inner position is $0 \leq v(n) \leq L(n-k)$. Elements of S_n consist of the number of processed messages per arrival time slot, such that processed messages can be original request messages if it

is the primary queue or retransmitted messages if it is the secondary queue. To obtain the number of original arrivals served for each processing time slot, the following conditional cases should be studied.

Case 1 If $M(n)+v(n) < L(n-k)$; the processing time slot for the next time slot $u(n+1)$ will not change (i.e. $u(n+1) = u(n)$), but only the inner position as $v(n+1) = v(n)+M(n)$. Then, the number of processed messages will be equal to $M(n)$, such that $S_n = \{s_n(n-k) = M(n)\}$.

Case 2 As the special case, if $M(n)+v(n) = L(n-k)$; then the server reaches to the next arrival time slot, i.e. $u(n+1) = n-k+1$, and the inner position for the next time slot will be the beginning of the next arrival time slot, i.e. $v(n+1) = 0$.

Case 3 In the case of the server has a remaining processing capacity after completion time slot $(n-k)$, which will be used for the next arrival time slot $(n-k+1)$, i.e. $L(n-k) < M(n)+v(n) < L(n-k)+L(n-k+1)$; processes can be calculated as follows:

$$\begin{aligned} u(n+1) &= n-k+1, \\ v(n+1) &= M(n)+v(n)+L(n-k+1), \\ S_n &= \{s_n(n-k) = L(n-k)-v(n), \\ &\qquad s_n(n-k+1) = M(n)+v(n)+L(n-k+1)\} \end{aligned} \tag{4.3}$$

Case 4 The special case of Eq. (4.3) is that if the remaining capacity after processing $L(n-k)-v(n)$ is equal to $L(n-k+1)$, then $u(n+1) = n-k+2$ and $v(n+1) = 0$.

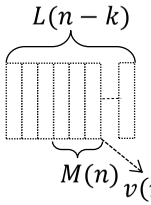
Case 5 As a generalization of the behavior, if the server using the capacity of $M(n)$ can process messages joined at arrival time slots up to m, for $m \leq n$, $u(n+1)$ becomes m, while $v(n+1)$ indicates an inner position of the time slot m. This can be formulated as follows:

$$\begin{aligned} u(n+1) &= m \\ v(n+1) &= M(n) - \left[L(n-k)-v(n)+\textstyle\sum_{i=n-k+1}^{m-1} L(i)\right] \\ S_n &= \{s_n(n-k) = L(n-k)-v(n), \\ &\qquad s_n(i) = L(i); \quad (where \quad i = (n-k+1),...,(m-1)), \\ &\qquad s_n(m) = v(n+1)\} \end{aligned} \tag{4.4}$$

Case 6 As a special case of Eq. (4.4), if the server processes all arrival messages from the queue (i.e. $m = n$), then $u(n+1) = n+1$ and $v(n+1) = 0$. In this case, the server will complete the work and have an idle time from the message processing point of view.

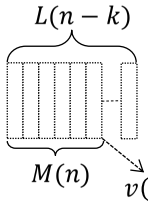
Figure 4.2 illustrates all cases of arrival time slot tracking process. For all equations, the elements of S_n consist of the number of processed messages per arrival time slot, such that processed messages can be original request messages if it is the primary queue or retransmitted messages if it is the secondary queue. This situation is denoted by using S_n^p and S_n^s, respectively, where specifically needed.

$M(n)+v(n)<L(n-k)$ **Case 1**



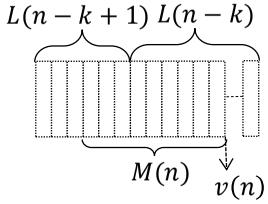
$u(n+1)=n-k$
$v(n+1)=v(n)+M(n)$
$S_n=\{s_n(n-k)=M(n)\}$

$M(n)+v(n)=L(n-k)$ **Case 2**



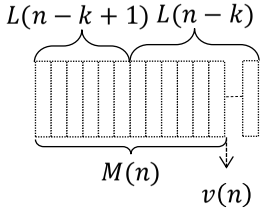
$u(n+1)=n-k+1$
$v(n+1)=0$
$S_n=\{s_n(n-k)=M(n)\}$

$L(n-k)<M(n)+v(n)<L(n-k)+L(n-k+1)$ **Case 3**



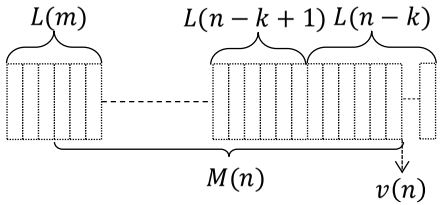
$u(n+1)=n-k+1$
$v(n+1)=M(n)+v(n)-L(n-k)$
$S_n=\begin{Bmatrix} s_n(n-k)=L(n-k)-v(n), \\ s_n(n-k+1)=M(n)+v(n)-L(n-k) \end{Bmatrix}$

$M(n)=L(n-k)-v(n)+L(n-k+1)$ **Case 4**



$u(n+1)=n-k+2$
$v(n+1)=0$
$S_n=\begin{Bmatrix} s_n(n-k)=L(n-k)-v(n), \\ s_n(n-k+1)=L(n-k+1) \end{Bmatrix}$

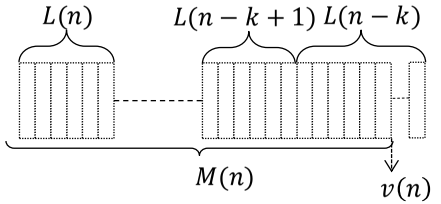
$\left[L(n-k)-v(n)+\sum_{i=n-k+1}^{m-1}L(i)\right]<M(n)<\left[L(n-k)-v(n)+\sum_{i=n-k+1}^{m}L(i)\right]$ $for\ m\le n$



Case 5

$u(n+1)=m$
$v(n+1)=M(n)-[L(n-k)-v(n)+\sum_{i=n-k+1}^{m-1}L(i)]$
$S_n=\begin{Bmatrix} s_n(n-k)=L(n-k)-v(n), \\ s_n(i)=L(i);for\ i=(n-k+1),\dots,(m-1), \\ s_n(m)=v(n+1) \end{Bmatrix}$

$\left[L(n-k)-v(n)+\sum_{i=n-k+1}^{n}L(i)\right]\le M(n)$ **Case 6**



$u(n+1)=n+1$
$v(n+1)=0$
$S_n=\{s_n(n-k)=L(n-k)-v(n),$
$s_n(i)=L(i);for\ i=(n-k+1),\dots,n\}$

Figure 4.2 : Illustration of cases of arrival time slot tracking process.

4.5 Modeling Removal Processes

By definition, the removal process removes all corresponding retransmissions from the secondary-queue when an original or non-redundant retransmission message is serviced from the queue, at time slot n. Therefore, there are two removal processes processing on the PRSM, namely the primary and secondary removal processes, denoted by W_n^p and W_n^s, respectively. W_n is used to represent the total effect of removal processes and defined as $W_n = W_n^p + W_n^s$.

In this section, we provide basic definitions of removal processes. The primary removal has almost a complete definition, such that the derivation of retransmission terms of Eq.(4.6) may differ among modeling cases, which will be given in the corresponding sections. The section for the secondary removal process consists only basic definitions, since the complete definition needs the representation of non-redundant retransmissions which is not defined yet.

4.5.1 Primary removal process

An original message may have i corresponding retransmitted messages, $0 \leq i \leq 6$, in the secondary-queue depending on how long the original message is delayed in the queue before served.

The primary removal process uses the arrival time slot tracking, descried in Section 4.4, for $L(n-k) = \lambda(n-k)$ and $M(n) = \mu(n)$. In this case, the set S_n consists of the amount of processed original arrival messages per arrival time slot, and denoted by S_n^p.

If the processing delay in the primary queue is greater than the retransmission timer T_j (i.e. $k > T_j$), the secondary queue may consist of redundant retransmitted messages corresponding to the processed original messages joined the queue at $(n-k)$, that will be removed by the removal process. These corresponding retransmitted messages can be represented by $r_j(n-k+T_j)$. The calculation of $r_j(n-k+T_j)$ may differ among used modeling cases.

Hence, by using the information of processed original messages from the primary queue provided by the arrival time slot tracking process at time slot n, the total number

of the removed corresponding redundant retransmissions from the secondary queue can be calculated as follows:

$$W_n^p = \sum_{i=n-k}^{n-k+|S_n^p|} w_n^p(i) \tag{4.5}$$

where, $|S_n^p|$ denotes the number of elements of the set S_n^p, and $w_n^p(i)$ denotes the total number of retransmission messages removed from the secondary queue due to processing on arrival time slot i, where $(n-k) \leq i \leq (n-k)+|S_n^p|$. Since an original message joined the primary queue at time slot $(n-k)$, may have j^{th}-time retransmissions in the secondary queue, $w_n(i)$ is calculated through six possible retransmissions:

$$w_n^p(i) = \sum_{j=1}^{6} min\left\{ s_n(i), \left[r_j(i+T_j) - \sum_{l=1}^{n} W_{n-l}^p \right] \right\} \tag{4.6}$$

where, the term $\sum_{l=1}^{n} W_{n-l}^p$ represents removals of the arrival time slot i during the previous primary queue processing through $(n-1)$, $(n-2)$, $\ldots$, 0. We use $min\{\}$ function since the server may process all arrivals joined at the time slot i or a part of them.

4.5.2 Secondary removal process

Secondary removal process is denoted by W_n^s. Because its complete definition differs among modeling cases, here we give the basic definition only. The complete definition is given in Section 4.6 and Section 4.8, for modeling of infinite buffer and network losses cases, respectively.

Whenever a non-redundant retransmission is serviced from the secondary queue, the secondary removal process removes all corresponding redundant messages from the secondary queue. A non-redundant retransmission, joined the queue as j^{th}-time retransmission at time slot $(n-k)$, may have up to $6-j$ corresponding redundant retransmissions in the secondary queue as depending on how long delayed in the queue before processing.

Because messages in the secondary queue are serviced by the remaining capacity $y(n)$ of the server, the secondary removal process is attached to $y(n)$. From definition, if the removal process is activated in the system, $y(n)$ shall not encountered with a redundant retransmission message when it picks a message for processing from the secondary

queue. Therefore, we have $M(n) = y(n)$. In this case, $L(n-k)$ will be equal to the aggregation of possible six non-redundant retransmissions, and the set S_n consists of the number of processed non-redundant retransmitted messages per arrival time slot and will be denoted as S_n^s. The elements of S_n^s are vectors consisting of the number of processed non-redundant retransmitted messages per retransmission sequence, such as 1^{st}-time retransmissions, 2^{nd}-time retransmissions, etc.

4.6 Model for Infinite Buffer Case

In this section, we first summarize the discrete-time fluid-flow model for retransmissions of the conventional SIP server for infinite buffer case [8]. Then, we derive the model of the PRSM for infinite buffer size case.

Because of infinite buffer size and lossless network assumption, there are no message losses in the server. Recalling definitions and parameters from Section 4.3, therefore, $\lambda(n) = \lambda^e(n)$ and $r(n) = r^e(n)$.

4.6.1 Model for conventional SIP with infinite buffer

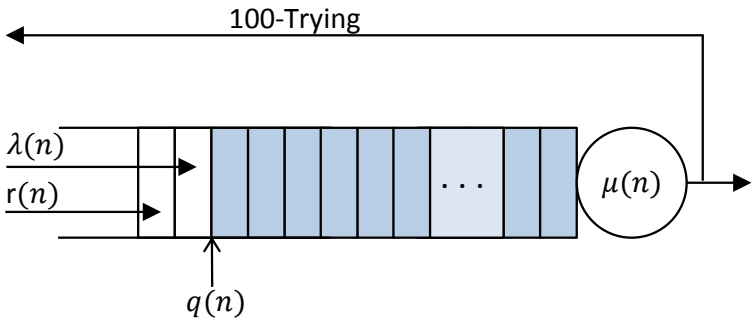


Figure 4.3 : Queuing model for the conventional SIP with infinite buffer.

In [8], the fluid-flow model for the conventional SIP server considers a typical SIP network which consists of an overloaded Proxy-server with infinite buffer and its multiple upstream originating servers.

4.6.1.1 Queuing model

Figure 4.3 depicts the considered queuing model for the conventional SIP server. Recalling basic definitions from Section 4.3, dynamics of the queuing model of the conventional SIP server are characterized by solely two processes; the arrival and

service processes. The arrival process is determined by an aggregation of original and retransmitted request message rates, denoted by $\lambda(n)$ and $r(n)$ for time slot n, respectively. The service process determined by the service time of an INVITE message denoted by $\mu(n)$. A service time of a message is measured starting from parsing a received INVITE message until a 100-Trying response sent. $\lambda(n)$ and $\mu(n)$ may have any arbitrary distributions.

The discrete-time queuing model uses difference equations to describe the queuing dynamics. At time slot n, $\lambda(n)+r(n)$ aggregated messages arrive the server while the queue size of the server is $q(n)$. The server processes $\mu(n)$ messages at time slot n. Hereby, the queue size at next time slot $(n+1)$ can be calculated as follows [8]:

$$q(n+1) = [q(n)+\lambda(n)+r(n)-\mu(n)]^{+} \tag{4.7}$$

where the notation $[x]^{+}$ is defined as $[x]^{+} \equiv max(0,x)$, since the queue size cannot be negative.

4.6.1.2 Modeling retransmissions

If the originating side (an upstream server or UAC) does not receive a response message within a specific time-out, it will trigger retransmission. There are six retransmissions at most for every original request message [1] (recall Eq. (4.1)).

To calculate j^{th}–time retransmissions, we need to note that, at time slot $(n-T_j)$, $\lambda(n-T_j)$ original request messages arrive, while the queue size is $q(n-T_j)$. From time slot $(n-T_j)$ to n, the server can process $\sum_{i=0}^{T_j}\mu(n-T_j+i)$ messages. The remaining portion of $\lambda(n-T_j)$ is retransmitted at time slot n such that the remaining portion cannot be greater than $\lambda(n-T_j)$. Hence, the number of j^{th}-time retransmitted messages at time slot n is:

$$r_j(n) = min\left\{\left[\lambda(n-T_j)+q(n-T_j)-\sum_{i=0}^{T_j}\mu(n-T_j+i)\right]^{+}, \lambda(n-T_j)\right\} \tag{4.8}$$

Note that; under the assumption that transmission and propagation delays are negligible, when the server processes a request message at time slot n, it sends a 100-Trying response to the originator which stops the retransmission timer for the corresponding request within the same time slot n. Therefore, the received retransmissions, at time slot n, do not contain messages corresponding to those processed messages.

4.6.1.3 Stability condition

Eqs.(4.7) and (4.8) (including Eq.(4.1)) indicate that the dynamic behavior of an overloaded SIP server has nonlinear characteristic. The model presented in [8] defines a stability condition as a function of initial queue size such that the server with a certain queue occupancy is stable if it can handle overload. The messages accumulated by a transient overload (e.g., a demand burst or a server slowdown) create a certain queue occupancy until the server returns to its normal service. Such transient queue occupancy may result in higher queuing delays, so redundant retransmissions bringing a server crash. According to the stability condition, the initial queue size $q(0)$ created by a demand burst can satisfy condition described by Eq.(4.9), and then the server is stable.

$$q(0) < min\Bigg\{(2^{(j+1)}-1)\mu T_1, \frac{\left((2^{(j+1)}+3\times 2^i - i - 4)\mu T_1 - ((i-1)2^i+1)\lambda T_1\right)}{(i+1)}, 1 \le j \le i \le 6\Bigg\} \tag{4.9}$$

For the Eq.(4.9), INVITE-"100-Trying" request-response pair with a deterministic arrival rate λ and a deterministic service rate μ are considered, and there are i retransmissions for the new original message. On the obtaining the Eq. (4.9), the main point for the stability condition is that the total incoming average rate should be less than the service rate. Assume that there would be i retransmissions for an arbitrary original INVITE request message, a conservative condition to maintain the stability will be indicated with the following equation:

$$(i+1)\lambda/\mu \le 1 \quad \Rightarrow \quad \mu \ge (i+1)\lambda \quad \Rightarrow \quad i \le (\mu-\lambda)/\lambda \tag{4.10}$$

To achieve the above sufficient stability condition, there is a need to guarantee that the original messages from both the initial queue size and the new arrivals are not retransmitted more than j times, where $j = \lfloor(\mu-\lambda)/\lambda\rfloor$. Then, Eq.(4.10) can be updated as follows:

$$i \le j = \left\lfloor \frac{(\mu-\lambda)}{\lambda} \right\rfloor \tag{4.11}$$

For example, $\lambda = 200$ messages/sec and $\mu = 1000$ messages/sec, Eq.(4.11) gives $j = 4$. Then, the stability condition of the overloaded conventional SIP server is calculated by using Eq.(4.9) as $q(0) < min\{15500, 8200, 6167, 5700, 6220\} = 5700$ messages.

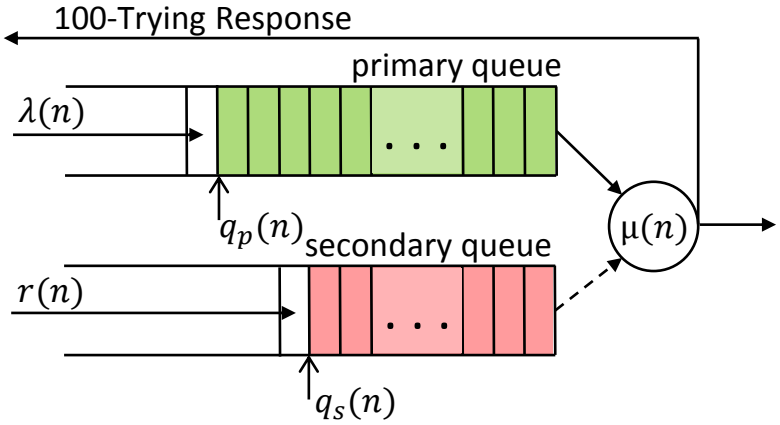


Figure 4.4 : The strict priority queuing model for the PRSM with infinite buffer.

Refer to [8] for the derivation of the stability condition equation, given with Eq.(4.9), and for evaluation of the behavior of the overloaded conventional SIP server under various initial queue size conditions. We also have conducted a test case to evaluate the behavior of SIP on the same initial queue size conditions in Section 6.1.1.

4.6.2 Model for PRSM with infinite buffer

4.6.2.1 Queuing model

Figure 4.4 shows the priority based queuing model of a SIP server with the PRSM and infinite buffer. The original INVITE requests with the arrival rate $\lambda(n)$ at time slot n, are placed into the primary queue and served with the FIFO manner. Retransmissions with the retransmission rate $r(n)$ at time slot n, are placed into the secondary queue and served with the FIFO manner only if the primary queue is empty. The priority mechanism is non-preemptive.

Compared to the conventional SIP given in Eq.(4.7), the primary queue size $q_p(n+1)$ at time slot $n+1$ for the PRSM excludes $r(n)$:

$$q_p(n+1) = [q_p(n) + \lambda(n) - \mu(n)]^+ \tag{4.12}$$

With the assumption of the strict priority, the server can process messages from the secondary queue only if the primary queue is empty. Therefore, the remaining capacity of the server at the time slot n is calculated as follows:

$$y(n) = -[q_p(n) + \lambda(n) - \mu(n)]^- \tag{4.13}$$

where $[x]^- \equiv min(0,x)$.

For the secondary queue size formulation, in addition to retransmission traffic, we need to consider both the remaining capacity of the server and removal process. At time slot n, $r(n)$ retransmission messages joins the secondary queue. Due to the processed original messages from the primary queue, the removal process removes corresponding redundant retransmissions from the secondary queue, such that an original message may have i retransmitted messages, $0 \leq i \leq 6$, in the secondary-queue depending on how long the original message is delayed in the primary queue before served. Besides, the remaining capacity of the server processes messages from the secondary queue. Thereby, the queue size of the secondary queue for the next time slot $(n+1)$ can be calculated recursively by subtracting the number of removed retransmissions by the removal process and the number of processed retransmissions by the remaining capacity of the server from the secondary queue size of the current time slot and adding the number of new retransmissions:

$$q_s(n+1) = [q_s(n) + r(n) - W_n - y(n)]^+ \tag{4.14}$$

where, W_n denotes the removal process.

Just in case, if the removal process is not applied, the size of the secondary queue at time slot $n+1$ can be obtained using the information at time slot n:

$$q_s(n+1) = [q_s(n) + r(n) - y(n)]^+ \tag{4.15}$$

4.6.2.2 Modeling retransmissions

Modeling the retransmissions for the PRSM case have similar characteristics to the conventional SIP server case. The distinction is that therms of j^{th}-time retransmissions, r_j for $1 \leq j \leq 6$, are calculated as based on the primary queue size, as follows:

$$r_j(n) = min\left\{ \left[\lambda(n-T_j) + q_p(n-T_j) - \sum_{i=0}^{T_j} \mu(n-T_j+i)\right]^+, \lambda(n-T_j) \right\} \tag{4.16}$$

4.6.2.3 Stability condition

Since transmission and propagation delays are negligible, and queuing and processing delays are dominant for retransmissions and buffer sizes for both primary and secondary queues are infinite (and hence there is no message loss in the network), all retransmissions are redundant as triggered by the excessive queuing delay and will be removed by the removal process of the PRSM. Thereby, the primary queue occupancy

is not affected from retransmitted messages. Under these circumstances, we have the following results for the stability condition of the PRSM.

Result 1 The stability condition of the PRSM is the same as the stability condition definition of a general queue with a single server and infinite buffer: $\rho = \lambda/\mu < 1$.

Result 2 Under the strict priority scheduling, the server remaining capacity process will always see the secondary queue as empty, and only the removal process will be effective at Eq.(4.14) to reduce the size of the secondary queue.

4.6.2.4 Retransmission removal processes

For the PRSM with infinite buffer size, the primary removal process is calculated by using Eq.(4.8) in Eq.(4.6) for time slot $(n-k+T_j)$. $r(n-k+T_j)$ can be calculated as follows:

$$r_j(n-k+T_j) = min\left\{ \left[\lambda(n-k)+q(n-k)-\sum_{i=0}^{T_j}\mu(n-k+i)\right]^+, \lambda(n-k) \right\} \quad (4.17)$$

As previously stated, since there is no non-redundant retransmissions for this case, the secondary removal process is never triggered. So, only the primary removal process is effective for this case. Thereby, $W_n = W_n^p$ for Eq.(4.14).

4.7 Model for Finite Buffer Case

In this section we derive the model for the PRSM with finite buffer size, under lossless network conditions. For the conventional SIP server, Hong, Y. et al. [9, 49] derived a discrete-time fluid-flow based analytical model of the conventional SIP server with finite buffer for tandem proxy cases. We derive the model for a single proxy SIP server as a reference model by simplifying the considerations such as ignoring the queuing of response messages from the downstream server. This derivation is given at Appendix A. Therefore, this section is considered only for the PRSM.

4.7.1 Queuing model

Figure 4.5 shows the queuing model of a single SIP server with the PRSM for the finite buffer capacity, where time is divided into small fixed-length discrete time slots. We

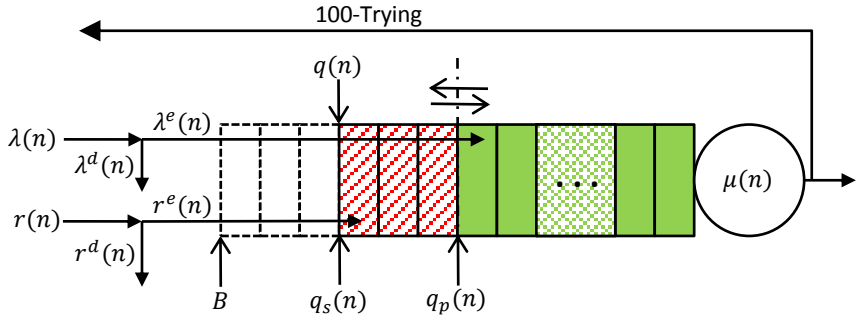


Figure 4.5 : The strict priority queuing model of the SIP server with PRSM for finite buffer case.

assume a single-queue representation as the combination of two queues which have a total buffer size of B. For the sake of presentation, we consider that there are two logical queues named primary and secondary, respectively. The total queue size $q(n)$ is equal to the summation of the primary queue size $q_p(n)$ and the secondary queue size $q_s(n)$.

By recalling basic definitions and parameters at Section 4.3, because of blocking characteristics of the model, the incoming arrivals including original and retransmitted messages are blocked and lost when the buffer is full (i.e., $q(n) = B$). Therefore, we need to calculate effective original, $\lambda^e(n)$, and retransmitted arrival rates, $r^e(n)$ per time slot. Consequently, for the formulation of queuing dynamics, assume $\lambda(n)$, $\mu(n)$, and B are given, the objective of the proposed model is to calculate $\lambda^e(n)$, $\lambda^d(n)$, $r^e(n)$, $r^d(n)$, $q_p(n)$, $q_s(n)$, $q(n)$ and other queuing dynamics such as queuing delays of both primary and secondary queues.

Initially, at the time slot n, the available buffer size is $B - q(n)$, and the server can process $\mu(n)$ request messages in this time slot. So, the buffer only allows the maximum of $B - q(n) + \mu(n)$ new request messages. Hence, the effective original arrival rate is:

$$\lambda^e(n) = min\{B - q(n) + \mu(n), \lambda(n)\}. \tag{4.18}$$

So, the number of dropped messages $\lambda^d(n)$ because of blocking can be calculated as

$$\lambda^d(n) = \lambda(n) - \lambda^e(n). \tag{4.19}$$

Under the enqueuing priority assumption, at the time slot n, the retransmissions enter the queue only if all of the original messages are joined the queue. In this case,

the buffer allows only maximum of $B - q(n) + \mu(n) - \lambda^e(n)$ retransmitted request messages. Hence, the effective retransmission rate at the current time slot n can be calculated as follows:

$$r^e(n) = min\{B - q(n) + \mu(n) + W_n - \lambda^e(n), r(n)\}. \tag{4.20}$$

where $W_n = W_n^p + W_n^s$. W_n^p and W_n^s denote primary and secondary removal processes, respectively. Then, dropped retransmissions $r^d(n)$ because of blocking can be calculated as

$$r^d(n) = r(n) - r^e(n). \tag{4.21}$$

The primary queue size of the next time slot $q_p(n+1)$ is determined by the effective original arrival rate joined the primary queue and the processing capacity of the server as it is limited by the buffer size B. Furthermore, if the queue is empty or the server still has the processing capacity, incoming new original request arrivals within the current time slot n are processed. So, $q_p(n+1)$ can be calculated as follows:

$$q_p(n+1) = \Big[min\{q_p(n) + \lambda^e(n) - \mu(n), B\}\Big]^+. \tag{4.22}$$

where $[x]^+ \equiv max(0,x)$, because the queue size cannot be negative.

According to the strict priority queuing discipline, the secondary queue is serviced if the primary queue is empty. The remaining processing capacity of the server, to process the secondary queue can be defined as follows:

$$y(n) = -\big[q_p(n) + \lambda^e(n) - \mu(n)\big]^- \tag{4.23}$$

where $[x]^- \equiv min(0,x)$. Consequently, the secondary queue size of the next time slot is:

$$q_s(n+1) = \Big[min\{q_s(n) + r^e(n) - W_n^p - W_n^s - y(n), B\}\Big]^+. \tag{4.24}$$

4.7.2 Modeling retransmissions

The arrival rate of retransmitted messages at time slot n is the aggregate rate of six retransmissions, since there are a maximum of six retransmissions for every original message according to the SIP standard. The original request messages which arrived at time slot $n - T_1$ will be retransmitted 1^{st}-time by originating UAs at time slot n if they cannot responded with 100-Trying by the server by time slot n, where T_1 is the

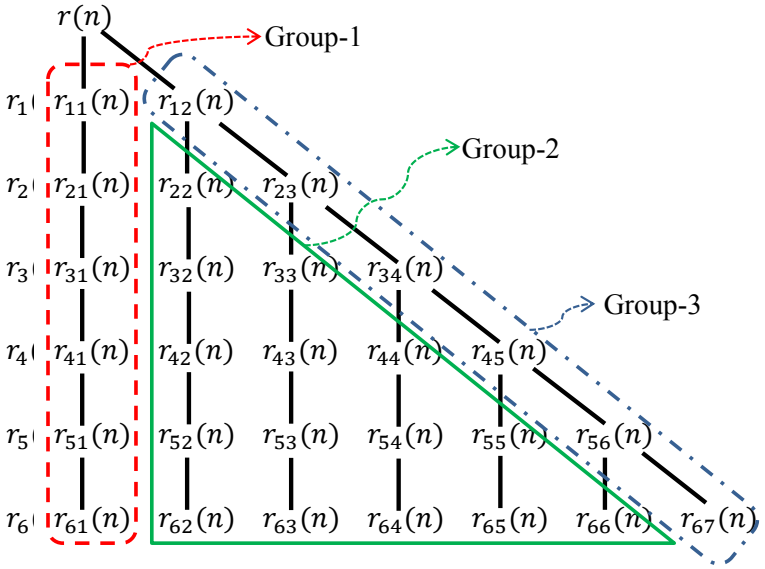


Figure 4.6 : Characterization of the retransmission messages.

first retransmission time defined in the SIP standard. Similarly, a message arrived at time slot $n - T_j$ will be retransmitted j^{th}-time by the originating UAs if there has not been a response is received by time slot n. Hence, the retransmission arrival rate, $r(n)$, at time slot n can be calculated by using Eq. (4.1).

Because of existence of non-redundant retransmissions, calculating $r_j(n)$ elements of $r(n)$ is more complicated when compared with the infinite case. Hence, to model $r(n)$, retransmissions can be decomposed into subcomponents, as it is proposed in [9] (see Figure 4.6), such that:

$$r_j(n) = \sum_{i=1}^{j+1} r_{ji}(n); 1 \leq j \leq 6. \tag{4.25}$$

As shown in Figure 4.6, for a better presentation we divided subcomponents into three groups. Group-1 and Group-2 consist of redundant retransmissions, while Group-3 consists of non-redundant retransmissions. Group-1 denotes j^{th}-time retransmissions caused by excessive delay of the original messages, $\lambda^e(n-T_j)$, joined the queue at time slot $(n-T_j)$. For example, r_{11} denotes the first time retransmissions of the effective original messages entered the queue at time slot $(n-T_1)$ and they are still in the queue at the current time slot n. They are transmitted because of their corresponding original messages are delayed for service. At time slot $(n-T_j)$, $\lambda^e(n-T_j)$ original request messages joined the queue while the queue size was $q_p(n-T_j)$. The server processes $\sum_{l=n-T_j}^{n} \mu(l)$ messages up to the current time slot n and the remaining

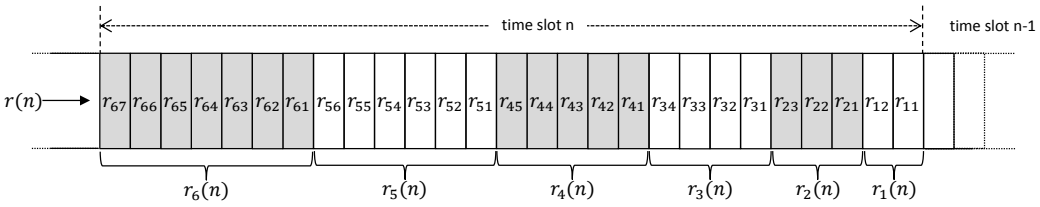


Figure 4.7 : An illustration of the enqueuing priority assumption for finite buffer case.

unprocessed messages are retransmitted at the current time slot n, such that the remaining unprocessed messages can be $\lambda^e(n-T_j)$ in max. So, subcomponents in Group-1 can be calculated as follows:

$$r_{j1}(n) = min\left\{ \left[\lambda^e(n-T_j) + q_p(n-T_j) - z_p(n-T_j,n)\right]^+, \lambda^e(n-T_j) \right\} \tag{4.26}$$

where $z_p(m,n) = \sum_{l=m}^{n} \mu(l)$.

According to the enqueuing priority assumption, within a time slot, the 1^{st}-time retransmitted messages have higher priority over the 2^{nd}-time retransmission on joining the queue. In this way, 2^{nd}-time retransmissions have higher priority over 3^{rd}-time retransmissions, and so on. More specifically, for the same level retransmissions, retransmitted messages because of queuing delay have higher priority over retransmissions due to lost requests (see Figure 4.7). For example, $r_{23}(n)$ has a lower priority compare to r_{11} and r_{12}. And $r_{23}(n)$ has a lower priority compare to r_{21} and r_{22}. Then the effective retransmission rates of Group-1 subcomponents can be calculates as follows:

$$r^e_{j1}(n) = min\left\{ \left[B + \mu(n) + W_n - q(n) - \lambda^e(n) - \sum_{l=1}^{j-1} r_l(n)\right]^+, r_{j1}(n) \right\} \tag{4.27}$$

where $W_n = W^p_n + W^s_n$ and the term $\sum_{l=1}^{j-1} r_l(n)$ represents higher priority subcomponents that joined the queue before $r_{j1}(n)$.

Subcomponents in the Group-3, which are represented as $r_{jj+1}, 1 \leq j \leq 6$, denote the number of messages that are being retransmitted j^{th}-time at time slot n while their corresponding original and all earlier retransmission messages have been dropped because of queue blocking. These retransmissions are non-redundant, since there are no corresponding messages of these requests in the queue. The first subcomponent, $r_{12}(n)$ is because of dropped original messages at time slot $(n-T_1)$, $\lambda^d(n-T_1)$. Similarly, the subcomponent r_{23} is related to the dropped part of the subcomponent r_{12}, such that $r_{23}(n) = r_{12}(n-T_2+T_1) - r^e_{12}(n-T_2+T_1)$, where $r_{12}(n-T_2+T_1)$

corresponds to original request messages arrived and dropped at time slot $(n-T_2)$ that retransmitted first time at $(n-T_2+T_1)$. In general, we can calculate components of Group-3 as follows:

$$r_{jj+1}(n) = r_{j-1j}(n-T_{jj-1}) - r^e_{j-1j}(n-T_{jj-1}) \tag{4.28}$$

where, $T_{ji} = T_j - T_i$ $(1 \leq i \leq j \leq 6)$, $r_{01}(.) = \lambda(.)$ and $r^e_{01}(.) = \lambda^e(.)$.

Subcomponents in Group-2, which are represented as $r_{ji}, 2 \leq i \leq j$, denote the number of messages that are being retransmitted j^{th}-time while their corresponding original and retransmitted messages earlier than $(i-1)^{st}$-time retransmissions have been blocked and their corresponding $(i-1)^{st}$-time retransmission has joined the queue at time slot $n-T_j+T_{i-1}$, but cannot be processed by the server until the time slot $n-T_j+T_i$. For example, retransmissions represented by $r_{ji}, 2 \leq i \leq j$, are because of processing delay of corresponding retransmissions joined the queue at $(i-1)^{st}$-time retransmission, which can be represented as r^e_{i-1i}; i.e. r_{22}, r_{32}, r_{42}, r_{52} and r_{62} correspond to the unprocessed part of r^e_{12}; r_{33}, r_{43}, r_{53} and r_{63} correspond to the unprocessed part of r^e_{23}, etc. Furthermore, the retransmissions represented by Group-2 are redundant, since, they are transmitted because of high processing delay on the corresponding Group-3 non-redundant retransmissions. As a result, subcomponents in Group-2 can be calculated as follows:

$$\begin{aligned} r_{ji}(n) = min\Big\{ & \big[r^e_{i-1i}(n-T_{ji-1}) + q_s(n-T_{ji-1}) + R_i(n-T_{ji-1}) \\ & - Z(n-T_{ji-1},n)\big]^+, \quad r^e_{i-1i}(n-T_{ji-1}) \Big\} \end{aligned} \tag{4.29}$$

where; $R_i(m) = \sum_{l=1}^{i-2} r^e_l(m) + \sum_{l=1}^{i-1} r^e_{i-1l}(m)$, which denote the aggregated retransmissions joined the queue prior to $r^e_{i-1i}(m)$ and $r^e_0(m) = 0$. Since the server processes and removes $Z(n-T_{ji-1},n)$ messages up to the current time slot n, unprocessed part of $r^e_{i-1i}(n-T_{ji-1})$, which may take values between 0 and $r^e_{i-1i}(n-T_{ji-1})$ will be retransmitted. Here, $Z(m,n)$ denote the total number of processed and removed messages from the secondary queue between the time slot m and n, such that $Z(m,n) = z_s(m,n) + h_p(m,n) + h_s(m,n)$, where $z_s(m,n) = \sum_{i=m}^{n} y(i)$, $h_s(m,n) = \sum_{i=m}^{n} W_i^{s*}$, $h_p(m,n) = \sum_{i=m}^{n} W_i^{p*}$ (where '*' indicates the members in the summation that do not contain removed retransmitted messages joined the queue after the time slot

m). $r^e_{i-1i}(n-T_{ji-1})$ can be calculated as follows:

$$r^e_{i-1i}(n-T_{ji-1}) = min\Big\{ \big[B - q(n-T_{ji-1}) - \lambda(n-T_{ji-1}) + \mu(n-T_{ji-1}) + W_{n-T_{ji-1}} - R_i(n-T_{ji-1})\big]^+, r_{i-1i}(n-T_{ji-1}) \Big\} \tag{4.30}$$

where; the term $r_{i-1i}(n-T_{ji-1})$, which belongs to the Group-3, can be calculated from Eq.(4.28). The effective part of $r_{ji}(n)$ can be calculated as follows:

$$r^e_{ji}(n) = min\Big\{ \big[B - q(n) - \lambda^e(n) + \mu(n) + W_n - R_i(n)\big]^+, r_{ji}(n) \Big\} \tag{4.31}$$

Up to this point, we derived the formulation of the basic fluid-flow model for SIP servers using the PRSM with finite buffer. Since the PRSM consists of removal processes, we will derive formulation for both primary and secondary queue removal processes in Section 4.7.3.

4.7.3 Removal processes

For the finite buffer case, are non-redundant retransmissions may occur in the server. Therefore, the secondary removal process is effective on the server, as well as the primary removal process. Following sections, we derive the formulation for the primary and secondary removal processes.

4.7.3.1 Primary removal process

As stated previously, the primary removal process removes all corresponding retransmissions from the secondary-queue when an original message is serviced from the primary-queue. An original message may have six retransmitted messages in the secondary-queue depending on how long the original message is delayed in the queue before served. Under the assumption that transmission delay is negligible, when the server processes a request message at the current time slot n, it sends a 100-Trying response to the originator which causes stop the retransmit timer. Therefore retransmitted messages from originators would be related to unprocessed messages rather than processed ones at the current time slot n.

The primary removal process uses the arrival time slot tracking, described in Section 4.4, for $L(n-k) = \lambda^e(n-k)$ and $M(n) = \mu(n)$. In this case, the set S_n consists of the amount of processed original arrival messages per arrival time slot and will

be denoted as S_n^p. If the processing delay in the primary queue is greater than the retransmission timer T_j (i.e. $k > T_j$), the secondary queue may consist of redundant retransmitted messages corresponding to the processed original messages joined the queue at $(n-k)$, that will be removed by the removal process. These corresponding retransmitted messages can be represented by $r_{j1}^e(n-k+T_j)$ and calculated by the following formula:

$$r_{j1}^e(n-k+T_j) = min\Bigg\{ \Big[B+\mu(n-k+T_j)+W_{n-k+T_j}-q(n-k+T_j) \\ -\lambda^e(n-k+T_j)-\sum_{l=1}^{j-1} r_l(n-k+T_j)\Big]^+, r_{j1}(n-k+T_j)\Bigg\} \tag{4.32}$$

In this way, removals, as a result of the primary queue processing, can be calculated with Eq.(4.5) by using the elements of S_n^p provided an output of the primary queue arrival time slot tracking process. We can re-write the Eq.(4.6) by using retransmissions belonging Group-1:

$$w_n^p(i) = \sum_{j=1}^{6} min\Bigg\{ s_n^p(i), \Big[r_{j1}^e(i+T_j) - \sum_{l=1}^{n} W_{n-l}^p\Big]\Bigg\}. \tag{4.33}$$

where $(n-k) \leq i \leq (n-k)+|S_n^p|$.

4.7.3.2 Secondary removal process

As stated previously, whenever a non-redundant retransmission is serviced from the secondary queue, the secondary removal process removes all corresponding redundant messages from the secondary queue. Because messages in the secondary queue are serviced by the remaining capacity $y(n)$ of the server, the secondary removal process is attached to $y(n)$. When the remaining capacity of the server processed a non-redundant retransmission belonging to the Group-3, the secondary removal process removes corresponding redundant retransmissions represented in the Group-2, if there are retransmissions have already joined the secondary queue.

As similar to the primary removal process, the secondary removal process uses the arrival time slot tracking. Because, at time slot n, an aggregation of Group-3 components can join the queue, $L(n-k)$ will be equal to the summation of these components, i.e. $L(n-k) = \sum_{j=1}^{6} r_{jj+1}^e(n-k)$. In this case $M(n) = y(n)$, and the set S_n consists of the number of processed non-redundant retransmitted messages per arrival time slot and will be denoted as S_n^s. The elements of S_n^s are vectors consisting of the

number of processed non-redundant retransmitted messages $r^e_{jj+1}(n-k)$; $(j=1,\ldots,6)$ joined the queue at time slot $(n-k)$. $r^e_{jj+1}(n-k)$ can be calculated recursively through Eq.(4.26) - Eq.(4.31), and specifically by using Eq.(4.30) for time slot $(n-k)$.

When a non-redundant retransmission joined the queue as j^{th}-time retransmission at time slot $(n-k)$ is serviced, if the processing delay in the secondary queue is greater than the next retransmission timer T_{j+1} (i.e. $k > T_{j+1}$), there may be up to $6-j$ corresponding redundant retransmissions joined the queue denoted by r^e_{mj+1}, where $m = j+1,\ldots,6$ (note that, there will be no a retransmission corresponding a non-redundant 6^{th}-time retransmission denoted as r_{67}). Because the non-redundant retransmission joined the queue at $(n-k)$ are corresponding to the original messages dropped at $(n-k-T_j)$, the corresponding redundant retransmissions have joined the queue at time slot $(n-k-T_j+T_m)$. Then the number of expected redundant retransmission, $r_{mj+1}(n-k-T_j+T_m)$, can be calculated recursively through Eq.(4.26) - Eq.(4.31) and specifically by using Eq.(4.29). The effective part $r^e_{mj+1}(n-k-T_j+T_m)$ can be calculated as similar to Eq.(4.31):

$$r^e_{mj+1}(x) = min\left\{ \left[B+\mu(x)+W_x-q(x)-\lambda^e(x)-R_m(x)\right]^+, r_{mj+1}(n-k-T_j) \right\} \tag{4.34}$$

where $x = n-k-T_j+T_m$ and $m = j+1,\ldots,6$.

Hence, the number of the removals by the secondary queue processing can be calculated as follows:

$$W^s_n = \sum_{i=n-k}^{n-k+|S^s_n|} w^s_n(i); \tag{4.35}$$

where

$$w^s_n(i) = \sum_{j=1}^{6} min\left\{ s^s_n(i), \left[\sum_{m=j+1}^{6} r^e_{mj+1}(i-T_j+T_m) - \sum_{l=1}^{n} W^s_{n-l} \right] \right\}. \tag{4.36}$$

In Eq.(4.36), the term $\sum_{l=1}^{n} W^s_{n-l}$ represents removals of the arrival time slot i during the previous secondary queue processing through $(n-1),(n-2),\ldots,0$.

4.8 Model for Lossy Network Conditions with Infinite Buffer Case

To the best of our knowledge, there is no a discrete-time fluid-flow model for the conventional SIP modeling retransmission traffic under lossy network conditions.

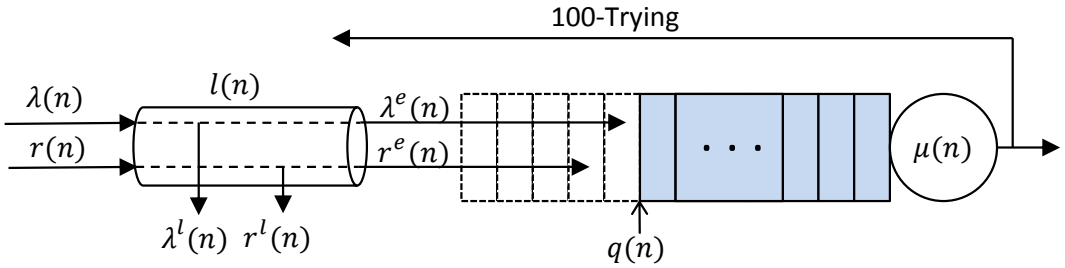


Figure 4.8 : Queuing model for the conventional SIP with infinite buffer under lossy network conditions.

Therefore, in this section, we derive the models of both the conventional SIP and the PRSM for lossy network conditions with infinite buffer case.

4.8.1 Model for conventional SIP

Figure 4.8 shows the queuing model of a single SIP server with infinite buffer in the lossy network conditions. Recalling basic definitions in Section 4.3, in addition to the arrival and service processes, dynamics of the queuing model of the conventional SIP server are characterized by loss rate, $l(n)$. Here, loss rate corresponds to the number of lost messages in each time slot. The arrival process is determined by an aggregation of original and retransmitted request message rates, denoted by $\lambda(n)$ and $r(n)$ for time slot n, respectively. The service process determined by the service time of an INVITE message denoted by $\mu(n)$. Loss rate, $l(n)$, includes losses from both $\lambda(n)$ and $r(n)$. It is denoted by $l_\lambda(n)$ when it is applied to the original arrivals, and by $l_r(n)$ when it is applied to retransmissions, such that number of lost message cannot be greater than the number of arrivals at time slot n. Thus, $\lambda^e(n)$ of $\lambda(n)$, which are called effective arrivals, can arrive the server because $\lambda^l(n)$ is lost. Similarly, $r^e(n)$ of $r(n)$ can arrive the server because $r^l(n)$ is lost. We assume no loss in responses. $q(n)$ denotes the queue size at time slot n.

Assuming that $\lambda(n)$, $\mu(n)$, and $l(n)$ are given and the aim is to calculate $\lambda^e(n)$, $\lambda^l(n)$, $r(n)$, $r^e(n)$, and queuing dynamics such as queue sizes and queuing delays. $\lambda(n)$, $\mu(n)$, and $l(n)$ may have arbitrary distributions.

At the time slot n, the network has a message loss rate $l(n)$. The original INVITE requests arrive with a rate $\lambda(n)$ and only effective arrival rate, $\lambda^e(n)$ arrived the server

while $\lambda^l(n)$ lost in the network. Then, we have the following calculations:

$$\lambda^e(n) = \lambda(n) - l_\lambda(n) \tag{4.37}$$

therefore,

$$\lambda^l(n) = l_\lambda(n). \tag{4.38}$$

Similarly,

$$r^e(n) = r(n) - l_r(n) \tag{4.39}$$

therefore,

$$r^l(n) = l_r(n) \tag{4.40}$$

According to the discrete-time fluid-flow modeling, the queue size at the next time slot $n+1$, $q(n+1)$ can be calculated from the queue size at the current time slot n, $q(n)$, by adding effective arrivals at the current time slot n, $\lambda^e(n)$ and $r^e(n)$, and subtracting the processing capacity of the server $\mu(n)$, as follows:

$$q(n+1) = [q(n) + \lambda^e(n) + r^e(n) - \mu(n)]^+ \tag{4.41}$$

where, $[x]^+ \equiv max(0,x)$, since the queue size cannot be negative.

To calculate effective retransmission rate $r^e(n)$, we need to know $r(n)$, which will be calculated in the following section.

4.8.1.1 Modeling retransmissions

Because this modeling case has non-redundant retransmission as similar to the finite case, we consider to use the triangle representation of decomposed retransmissions described in 4.7.2, by implying that non-redundant retransmissions are because of loss, instead of blocking, such that; subcomponents in the Group-3, which are represented as $r_{jj+1}, 1 \leq j \leq 6$, denote the number of messages that are being retransmitted j^{th}-time at time slot n while their corresponding and all earlier retransmission messages have been lost in the network (refer to Section 4.7.2 for details). Therefore, only the calculation of effective subcomponents will be different. Then, Eq.(4.1) and Eq.(4.25) are still valid for the conventional SIP case and decomposed retransmissions can be calculated through the following formulas recursively:

$r_{j1}(n)$ can be calculated by using Eq.(A.3). Then the effective retransmission rates of Group-1 subcomponents can be calculates as follows:

$$r^e_{j1}(n) = r_{j1}(n) - l_{r_{j1}}(n) \tag{4.42}$$

Eq.(4.28) can be used to calculate $r_{jj+1}(n)$ for the conventional SIP case.

$r_{ji}(n)$ can be calculated by using Eq.(A.5). Effective retransmissions for $r^e_{i-1i}(n - T_{ji-1})$ required by Eq.(A.5) can be calculated as follows:

$$r^e_{i-1i}(n - T_{ji-1}) = r_{i-1i}(n - T_{ji-1}) - l_{r_{i-1i}}(n - T_{ji-1}) \tag{4.43}$$

The effective portion of $r_{ji}(n)$ can is calculated as follows:

$$r^e_{ji}(n) = r_{ji}(n) - l_{r_{ji}}(n) \tag{4.44}$$

4.8.2 Model for PRSM

Figure 4.9 shows the queuing model of a single SIP server with the PRSM for infinite buffer capacity and in the lossy network conditions. Because of infinite buffer size, the queuing model consists of a double-queue representation, for the primary and secondary queues, respectively. By recalling basic definitions and parameters at Section 4.3, all the six processes are applicable for this model. Because of lossy network of the model, the incoming arrivals including original and retransmitted messages are lost according to loss rate, $l(n)$, where loss rate corresponds to the number of lost messages in each time slot. It is denoted by $l_\lambda(n)$ when it is applied to the original arrivals, and by $l_r(n)$ when it is applied to retransmissions, such that number of lost message cannot be greater than the number of arrivals at time slot n. Thus, $\lambda^e(n)$ of $\lambda(n)$, which are called effective arrivals, can arrive the primary queue of the server because $\lambda^l(n)$ is lost. Similarly, $r^e(n)$ of $r(n)$ can arrive the secondary queue of the server because $r^l(n)$ is lost. We assume no loss in responses.

Assuming that $\lambda(n)$, $\mu(n)$, and $l(n)$ are given and the aim is to calculate $\lambda^e(n)$, $\lambda^l(n)$, $r(n)$, $r^e(n)$, and queuing dynamics such as queuing dynamics such as queue sizes of both primary and secondary queues and their queuing delays. $\lambda(n)$, $\mu(n)$, and $l(n)$ may have arbitrary distributions.

At time slot n, $\lambda^e(n)$, $\lambda^l(n)$, $r^e(n)$ and $r^l(n)$ can be calculated equations through Eq.(4.37)-Eq.(4.40). The primary queue size of the next time slot $q_p(n+1)$ is

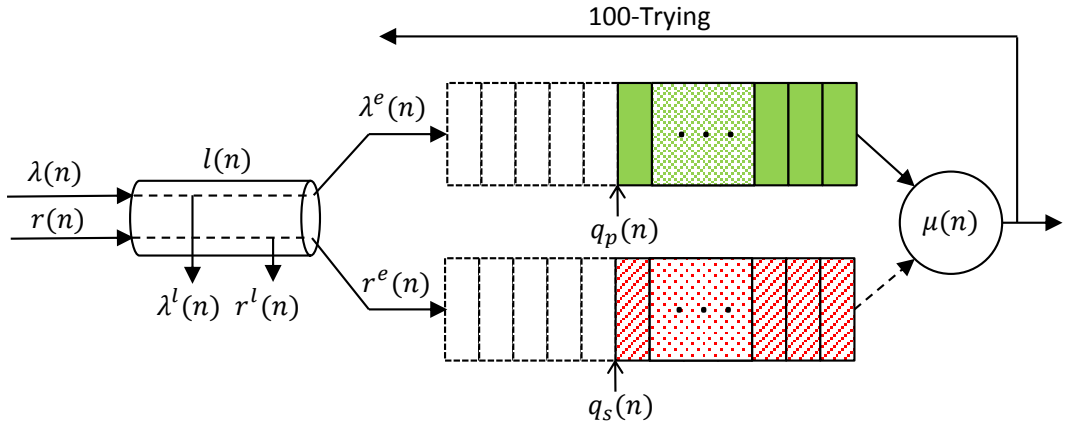


Figure 4.9 : The strict priority queuing model for the PRSM with infinite buffer under lossy network conditions.

determined by the effective original arrival rate joined the primary queue and the processing capacity of the server. Furthermore, if the queue is empty or the server still has the processing capacity, incoming new original request arrivals within the current time slot n are processed. So, $q_p(n+1)$ can be calculated as follows:

$$q_p(n+1) = \left[q_p(n) + \lambda^e(n) - \mu(n)\right]^+. \tag{4.45}$$

where $[x]^+ \equiv max(0,x)$, because the queue size cannot be negative.

Effective retransmissions join the secondary queue. Remaining service capacity. $y(n)$, and removal processes are effective in the calculation of the secondary queue size. The remaining service capacity, $y(n)$, can be calculated by using Eq.(4.23). Consequently, the secondary queue size of the next time slot is:

$$q_s(n+1) = \left[q_s(n) + r^e(n) - W_n^p - W_n^s - y(n)\right]^+. \tag{4.46}$$

where W_n^p and W_n^s are the primary and secondary removal processes respectively. We derived the formulation of removal processes in Section 4.8.2.2. To calculate the effective retransmissions, $r^e(n)$, we need the calculate $r(n)$. This formulation is derived in the following Section 4.8.2.1.

4.8.2.1 Modeling retransmissions

Because this modeling case has non-redundant retransmission as similar to the finite case, we consider to use the triangle representation of decomposed retransmissions described in 4.7.2, by implying that non-redundant retransmissions are because of loss,

instead of blocking, such that; subcomponents in the Group-3, which are represented as $r_{jj+1}, 1 \leq j \leq 6$, denote the number of messages that are being retransmitted j^{th}-time at time slot n while their corresponding and all earlier retransmission messages have been lost in the network (refer to Section 4.7.2 for details). Therefore, only the calculation of effective subcomponents will be different. Then, Eq.(4.1) and Eq.(4.25) are still valid for the PRSMP case and decomposed retransmissions can be calculated through the following formulas recursively:

$r_{j1}(n)$ can be calculated by using Eq.(4.26). Then the effective retransmission rates of Group-1 subcomponents can be calculated by using Eq.(4.42).

Similarly, components of Group-3, $r_{jj+1}(n)$ and r_{ji} can be calculated by using Eq.(4.28) and Eq.(4.29), respectively, while $r^e_{i-1i}(n-T_{ji-1})$ can be calculated by using Eq.(4.43).

The effective portion of $r_{ji}(n)$ can is calculated by using Eq.(4.44).

4.8.2.2 Removal processes

The number of retransmissions removed by the primary can be calculated as defined in Section 4.7.3.1, except replacing Eq.(4.32) as follows:

$$r^e_{j1}(n-k+T_j) = r_{j1}(n-k+T_j) - l_{r_{j1}}(n-k+T_j) \tag{4.47}$$

Similarly, the number of retransmissions removed by the secondary removal process can be calculated as defined in Section 4.7.3.2, except replacing Eq.(4.34) as follows:

$$r^e_{mj+1}(x) = r_{mj+1}(n-k-T_j) - l_{r_{mj+1}}(m)) \tag{4.48}$$

where $x = n-k-T_j+T_m$ and $m = j+1,\ldots,6$.

4.9 Performance Metrics

In this section, we will derive the formulation of SIP related performance metrics, for all model cases, to evaluate the performance improvements of the PRSM compared to the conventional SIP. The *failed call attempt* metric represents the ratio of the number of INVITE requests that cannot receive an 100-Trying response within the request timeout timer, $64 \times T_1$, to the number of the total call attempt. The *response delay*

metric measures the duration between the transmission of original INVITE requests and the reception of the corresponding 100-Trying responses.

4.9.1 Failed call attempts

According to the SIP RFC 3261, retransmissions cease when a response is received or the request timeout timer expires which has a default value of $64 \times T_1$. The request timeout timer is called Timer B, T_B for INVITE transactions. A call attempt fails upon the expiry of T_B.

For infinite buffer case, simply non-processed portion of original arrivals that joined the queue at time slot $(n - T_B)$ that is still in the queue at time slot n gives the number of the failed call attempts. It can be calculated as similar to Eq. (4.8):

$$f(n) = min\left\{ \left[\lambda(n-T_B) + q_p(n-T_B) - \sum_{i=0}^{T_B} \mu(n-T_B+i)\right]^+, \lambda(n-T_B) \right\} \tag{4.49}$$

Note that, Eq. (4.49) can also be used for the conventional SIP by using $q(n)$ instead of $q_p(n)$.

For the finite buffer case, to calculate failed call attempts we need to consider partially blocked requests, which are crating non-redundant retransmissions, and fully blocked requests that never joined the queue for T_B duration, as well as excessively delayed original requests. Hence, at time slot n, the number of the failed call attempts can be calculated as follows:

$$f(n) = r_{b1}(n) + \sum_{i=1}^{6} r_{bi+1}(n) + r_{67}^{d}(n) \tag{4.50}$$

where $r_{b1}(n)$ represents non-processed part of original arrivals that joined the queue at time slot $(n - T_B)$, $\lambda^e(n - T_B)$, and they are still in the queue at time slot n:

$$r_{b1}(n) = min\left\{ \left[\lambda^e(n-T_B) + q(n-T_B) - z(n-T_B, n)\right]^+, \lambda^e(n-T_B) \right\} \tag{4.51}$$

The members of summation represent retransmissions that their corresponding original messages and all earlier retransmissions have been blocked until the i^{th}-retransmission and some of them are still in the queue, at T_B timer expiry. These members can be computed by using Eq.(4.29). The last term, $r_{67}^{d}(n)$ represents that the part of original messages arrived at $(n - T_B)$ and cannot join the queue for all retransmission attempts,

hence it can be calculated as:

$$r_{67}^{d}(n) = r_{67}(n) - r_{67}^{e}(n) \tag{4.52}$$

$r_{67}(n)$ can be calculated with Eq.(4.28) for $j = 6$.

Eqs. (4.50) and (4.52) can be also used for the conventional SIP. For the calculation of the members of summation Eq. (A.5) is used.

For the lossy network condition with the infinite buffer case, we need to re-write the Eq.(4.50) to reflect loss situation instead of drop because of blocking:

$$f(n) = r_{b1}(n) + \sum_{i=1}^{6} r_{bi+1}(n) + r_{67}^{l}(n) \tag{4.53}$$

where $r_{67}^{l}(n)$ represents that the part of original messages arrived at $(n - T_B)$ and lost in the network for all retransmission attempts, hence it can be calculated as:

$$r_{67}^{l}(n) = r_{67}(n) - r_{67}^{e}(n) \tag{4.54}$$

4.9.2 Response delay

Response Delay is defined as the duration between the time that the original request message arrives the server and the time that the corresponding 100-Trying response sent to the sender. Under the assumption that transmission delay is negligible, this will be almost equal to the delay between the transmission of the INVITE request and the reception of the corresponding 100-Trying response by the SIP client. We assume that the server sends the response as soon as after parsing the request message and then continue the further processing of the request.

According to RFC 3261, the SIP client transaction state, which is receiving a 100-Trying response to an INVITE message, makes a transition from the *"Calling"* state to the *"Proceeding"* state and cease further retransmissions. Then, any further 100-Trying responses, which are the most likely due to processing of corresponding retransmissions, will be passed to TU. We considered that these further 100-Trying responses corresponding to a request are redundant and will be discarded by the TU. Hence, we consider that the first 100-Trying response stops the timer for measuring the response delay. While in the case of infinite buffer and lossless network conditions, the original request message is the first responded message, in the case of finite

buffer and/or lossy network conditions, because original messages and all earlier non-redundant retransmissions are likely to be blocked or lost, the first responded request message may be the first non-redundant retransmission that joined the queue of the server. Note that, because the removal process of the PRSM removes all redundant retransmissions from the queue due to the processing of the corresponding original or non-redundant retransmitted message, no redundant 100-Trying responses will occur in the case of the PRSM with the removal process.

The computation of the response delay for he PRSM uses arrival time slot tracking, for both the primary and the secondary queues, which provide the information about the time difference between the arrival and the service starting times of a request, in slotted time manner. As previously given, with a general representation, for the arrival time slot tracking of the primary queue with settings $L(n-k) = \lambda^e(n-k)$ and $M(n) = \mu(n)$, the elements of the set S_n^p will consist of the amount of processed original request messages per time slot, and the total delay on the primary queue processing. When a message arrived at time slot $n-k$ is serviced at time slot n, it will be delayed k time slots for receiving 100-Trying response. Consequently, the delay for time slot n can be calculated as follows:

$$D_n^p = \sum_{i=k}^{k+|S_n^p|} s_n^p(n-i) \times i \tag{4.55}$$

Note that, for the infinite buffer case with lossless network assumption, the setting for the arrival time slot tracking will be $L(n-k) = \lambda^e(n-k)$ because $\lambda(n) = \lambda^e(n)$.

Similarly, for the arrival time slot tracking of the secondary queue with settings $L(n) = \sum_{j=1}^{6} r_{jj+1}^e(n)$ and $M(n) = y(n)$ and the elements of the set S_n^s will consist of the number of processed non-redundant retransmitted messages, denoted with $r_{jj+1}^e(n)$; $(j = 1, \ldots, 6)$, so the total delay on the secondary queue processing for the time slot n can be calculated as follows:

$$D_n^s = \sum_{i=k}^{k+|S_n^s|} \sum_{j=1}^{6} s_n^s(n-i) \times (i+T_j) \tag{4.56}$$

The total response delay for PRSM at the time slot n is calculated as:

$$D_n^{PRSM} = D_n^p + D_n^s \tag{4.57}$$

Note that, for the infinite buffer case with lossless network assumption, because there is no non-redundant retransmissions in the secondary queue, the secondary removal process is never triggered. Therefore, D_n^s always will be zero for this modeling case.

The response delay can also be calculated for the conventional SIP server by using the arrival time slot tracking. For the finite buffer and lossy network condition cases, the arrival time slot tracking will has settings $L(n-k) = \lambda^e(n-k) + r^e(n-k)$ and $M(n) = \mu(n)$. In this case, the elements of S_n will consist of processing counts from both redundant and non-redundant retransmissions as well as original messages (such that $S_n = \{s_n(n-k) = \lambda^e(n-k) + r^e(n-k), s_n(n-k+1) = \lambda^e(n-k+1) + r^e(n-k+1), \dots\}$). It is clear that, the response delay computing shall not include redundant retransmissions, since they were previously responded during processing of their original or non-redundant retransmit messages and they will be discarded by the request sender. Therefore, only non-redundant effective retransmissions of S_n are considered in the computing of the response delay, which are denoted as r^e_{jj+1} and calculated as given at [9]. Consequently, the response delay for the conventional SIP case, D_n^c, for time slot n can be calculated as follows:

$$D_n^c = \sum_{i=k}^{k+|S_n|} \sum_{j=0}^{6} s_n^*(n-i) \times (i+T_j); T_0 = 0 \tag{4.58}$$

where, '*' is used to indicate that the term $s_n^*(n-i)$ is the part of $s_n(n-i)$ which consists only original messages and non-redundant retransmissions.

For the infinite buffer case, with $\lambda(n) = \lambda^e(n)$ and $r(n) = r^e(n)$, the settings of the time slot tracking will be $L(n-k) = \lambda(n-k) + r(n-k)$ and $M(n) = \mu(n)$. Then the response delay is calculated by using Eq.(4.58) as based on the processing information for original messages, since all retransmissions are redundant.

5. RETRANSMISSION ANALYSIS AND SIMULATION TOOL

To fully understand the behavior of SIP server under various traffic loadings, it is necessary to use an exploratory analysis tool which is capable of generating realistic arrival and departure patterns and controlling various SIP server parameters with reporting capabilities. In this chapter, we present a discrete-event based analysis tool which can be used to evaluate the effect of SIP retransmission mechanism on the SIP server under various scenarios as well as to evaluate the performance of the PRSM.

The traffic model of the tool uses Markov Modulated Poisson Process (MMPP) which is heavily used to model bursty arrivals in packet based telecommunication networks [50–54]. In a typical MMPP traffic model, different exponential arrival rates are used corresponding to various MMPP arrival modes. MMPP can also be used for modeling different departure rates of the server such as normal message processing and maintenance modes [55, 56]. An accurate analysis and simulation tool should consider both demand burst and diversity on the server processing capability (e.g., a registration request requires more processing power than an INVITE request since it needs to access a database for user credential verification). The avalanche restart phenomena described in RFC 5390 can be realized by creating the server overload situation using variations in both arrivals and departures. The tool, we presented in this section, is capable of generating various arrival and departure patterns corresponding to both cases.

The tool has the capability of enabling and disabling the retransmissions that can be used to observe queuing dynamics causing retransmissions. It supports various queuing models such that single or prioritized multiple queues, with infinite buffer size(s) or configurable finite buffer size(s). Multiple virtual queues may share a common buffer or each virtual queue may have its own separated buffer. The tool is also capable of simulating lossy network conditions. The tool, which is implemented in Java, can be run in systems where Java Virtual Machine (VM) is available. In

the current version, the tool is used through command-line terminal and the GUI (Graphical User Interface) will be added as a future work.

In this chapter, we present experiments to observe the behavior of the retransmission mechanism using various MMPP based traffic arrivals and departures which are generated independently from the retransmission process. The mode durations of MMPP and the buffer size of the SIP server are varied to evaluate the effect of SIP retransmissions on the SIP server overloading. Numerical results indicate that the number of total retransmissions significantly decreases if an incoming request is dropped instead of buffered when the queuing delay of the server is equal or greater than the first retransmission timer.

This chapter is organized as follows. Section 5.1 provides a brief background on MMPP. Section 5.2 presents the functional architecture of the tool while Section 5.3 provides the software architecture. The performance evaluation of the tool is presented in Section 5.4 to verify the accuracy of the tool with respect to theoretical expectations of MMPP. This section also present a set of experiments to evaluate the effect of retransmissions on a SIP server. Finally, the performance evaluation of the PRSM is presented in Section 5.5.

5.1 MMPP Background

MMPP consists of multiple Poisson processes modulated by a Markov process. The modulation mechanism defines modes (states) [57] that in a mode i of m modes, the rate parameters of the Poisson process are different from those of Poisson processes in other modes (states). Transition between modes are governed by an underlying continuous-time Markov chain.

MMPP is heavily used to model bursty arrivals in packet based telecommunication networks [50–54]. It can also be used for modeling different departure rates: (i) a server that processes jobs from different sources [56], (ii) system breakdowns, where none of the servers work and partial failures, where all servers work with lower efficiency [58], and (iii) a server working in different modes such as normal mode and maintenance mode [55].

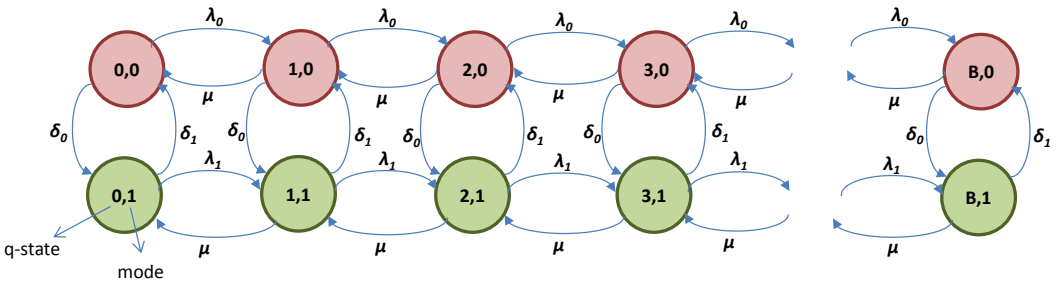


Figure 5.1 : State transition diagram of MMPP(2)/M/1/B queue.

In a typical MMPP traffic model, different exponential arrival rates can be used corresponding to various MMPP arrival modes. For example, in the case of two modes of MMPP (MMPP(2)), to create a bursty traffic arrival behavior, two arrival rates can be used to represent overloading and non-overloading scenarios, respectively. In overloading mode, the model uses a Poisson process with an INVITE request arrival rate λ_0 for a period of time that is exponentially distributed with parameter δ_0. Then, it moves to non-overloading mode, in which it behaves as a Poisson process with arrival rate λ_1 for a period of time that is exponentially distributed with parameter δ_1, where δ_0 and δ_1 represent mode duration parameters. Figure 5.1 is an example of MMPP/M/1/B queue modeling a finite buffer single-server queue with MMPP(2) arrival and exponential service rates. In this case, the average arrival rate can be calculated as follows [57, 59]:

$$\lambda_{av} = \frac{\delta_1}{\delta_0 + \delta_1}\lambda_0 + \frac{\delta_0}{\delta_0 + \delta_1}\lambda_1. \tag{5.1}$$

Then, in a queuing model with infinite buffer, the stability condition is $\rho = (\lambda_{av}/\mu) < 1$. Thus, for given λ_0, λ_1 and μ, mode transition parameters (i.e., δ_0 and δ_1 for MMPP(2)) determines the stability condition. For example, in overloading mode, the arrival rate, λ_0 can fill the queue with $\rho = (\lambda_0/\mu) > 1$ and the following non-overloading mode will be able to clear the queue with $\rho = (\lambda_0/\mu) < 1$. In this case, the traffic is called as bursty. In general, since the mode duration parameters determine the time of staying in each mode, smaller values of δ provide higher mean queue size and blocking probability [57]. For example, system stays in the demand burst mode with the mean time $1/\delta_0 = 1$ seconds and in the normal traffic mode with the mean time $1/\delta_1 = 0.1$ seconds. For $1/\delta_0 = 1$, higher arrival rate in demand burst

mode for a sufficiently long time builds up a long queue and/or blocked packets causing retransmissions.

MMPP traffic model is analytically tractable and state probabilities can be calculated for both queuing systems with infinite [50, 55] and finite buffer [57, 59]. Then, once state probabilities are calculated, the blocking probability of the queuing model can also be calculated [57]. State probabilities of an MMPP(2)/M/1/B queue is calculated by the infinitesimal generator $2B \times 2B$ matrix. In [57], Gauss-Seidel, successive approximation method is suggested for solving the steady-state equations. A software program, written in Java, using this method is given in [60].

Similarly, in a typical MMPP service model, different exponential service rates can be used corresponding to various MMPP service modes. For example, two service rates μ_0 and μ_1 are used corresponding to two different MMPP service modes, such that for a single arrival mode, the queuing model can be represented as M/MMPP(2)/1. In overloading (non overloading) mode, the model uses an exponentially distributed service time $\mu_0(\mu_1)$ for a period of time that is exponentially distributed with parameter $\delta_0(\delta_1)$. We derived the infinitesimal generator matrix for M/MMPP(2)/1/B queue in App. B. A software program, in Java, solving the steady-state equations according to Gauss-Seidel successive approximation method is available in [61].

5.2 Functional Architecture

Figure 5.2 depicts the high-level functional blocks of our tool. *Event Manager*, which behaves as an origination node, identifies each arrival event (which simulates an INVITE message), provides it to *Queue Simulation*. To simulate hop-by-hop SIP behavior, *Queue Simulation* provides the information whenever an arrival event is pulled from the queue for the service. This simulates 100-Trying response. The distribution of arrival events can be configured as deterministic, Poisson or MMPP. For MMPP traffic mode configuration, the *Event Manager* needs the number of modes and mode duration values for each mode to be configured as well as arrival rates per mode. The configuration of arrival rates determines an *Arrival Model* for the *Event Manager*.

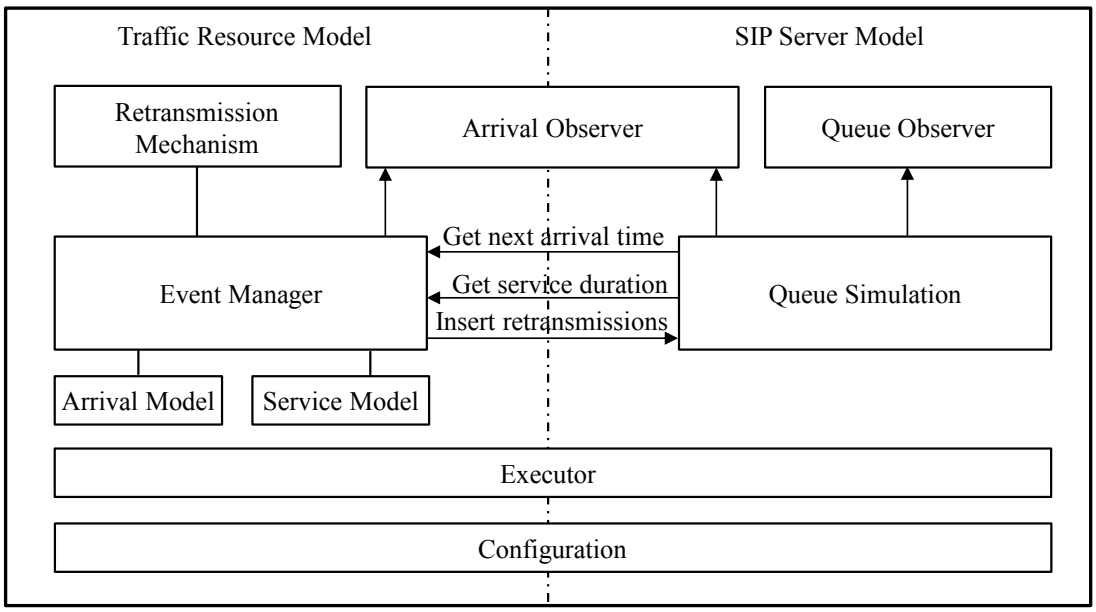


Figure 5.2 : Analysis and Simulation Tool Functional Architecture.

Event Manager also provides a service duration to *Queue Simulation* for each arrival to be processed. The distribution of service durations can be configured as deterministic, exponential or MMPP. Therefore, for MMPP service mode configuration, *Event Manager* needs the number of modes and mode duration values for each mode to be configured as well as service rates per mode. Similarly, the configuration of service rates determines a *Service Model* for *Event Manager*. Both *Arrival Model* and *Service Model* can be configured as MMPP. There is no restriction on the number of modes for both models.

For each arrival event provided to *Queue Simulation*, *Event Manager* starts a timer with T_1 value for a possible retransmission. When *Event Manager* receives the information about the processing of arrival events from *Queue Simulation*, it stops the corresponding retransmission timer as well. To prevent slowdown of the software because of a need for processing a large number of retransmission timers, a two-container structure, which consists of a hast table and a doubly-linked list, is used for the retransmission timing. The hash table is used to access timers efficiently in the case of removing them when 100-Trying event is received. The doubly linked list provides a sorted list according to expiry time. The hash table is also used to find appropriate position for newly inserted timers into the list. This reduces the insertion time into the list.

Because of the discrete-event nature of the implementation, changes in system state variables are associated with events that occur at discrete time instances only. Therefore, *Event Manager* checks for any retransmission timer expiry when *Queue Simulation* requires a next arrival event. The time assigned for this newly produced arrival event is considered as the *current-time* of the system. If there are timers in the retransmission timer list that older than the *current-time*, a retransmitted arrival event is created per timer and provided to *Queue Simulation* as an (retransmitted) arrival event before the newly created arrival event. For each retransmitted arrival event, *Event Manager* starts a new retransmission timer according to the exponential backoff algorithm of SIP retransmission mechanism. Thus, SIP retransmission mechanism is simulated in this way.

The retransmission mechanism can be disabled by the configuration, but *Event Manager* still starts the first retransmission timer for arrival events. This is called "observer mode" that retransmission messages are not sent although their corresponding timers are expired but only detection and counting. Since the MMPP models do not consider the retransmissions, the observer mode can be to verify that our implementation provides consistent behavior with analytical solutions based on MMPP. The case that retransmissions are applied to the *Queue Simulation* is called "applier mode" and simulates SIP retransmission behavior such as SIP INVITE request is retransmitted upon no 100-Trying response received from the terminating node.

Event Manager can also simulate lossy network conditions according to the user specified loss rate. It is considered that, loss rate is configured as a percentage of total received arrivals. In the current implementation, uniformly distributed loss behavior is considered such that for each arrival event created for *Queue Simulation*, if the number obtained from the uniform distribution, $U(0,1)$, is in range of the configured percentage, that arrival is considered as lost in the network. But corresponding retransmission time is still started for applying the retransmission behavior. This algorithm is also applied for retransmitted arrival events, but not 100-Trying responses.

The implementation of *Queue Simulation* is inspired by the discrete-event driven M/M/1 queuing simulation in [62]. M/M/1 queuing simulation is modified to support priority queue behavior as well as finite queue behavior. In the case of infinite buffer size, the queue size is restricted by the memory resources of the system. According to

the provided configuration, the appropriate queue type run as the *Queue Simulation*. Then it processed events according to the determined queuing schedule. In the current implementation, there are three types of priority queues: *(i)* new arrivals have higher priority over retransmission, *(ii)* retransmissions have higher priority over new arrivals, and *(iii)* priority levels separately for retransmissions according to the retransmission sequence, while new arrivals have highest priority. In the case of finite buffer case, all priority queues share the buffer size. In the case of blocking, it is considered that an arrival event is just dropped without further response to the originating side.

Queue Observer periodically collects queue length information according to PASTA (Poisson Arrivals See Time Averages) property. *Arrival Observer* observers call attempts by creating arrival objects for each new arrival and observing it throughout its life (i.e. until its processing completed in the queue). It collects all information about arrivals, such as retransmission status, loss status, dropping status because of blocking, service starting time and service completion time. *Executor* is responsible for parsing the configuration file, creating required software components accordingly and starting the simulation. It can be triggered from a command-line interface or GUI. *Configuration* uses text-based files consists of key/value paired parameters to configure the tool before start.

5.3 Software Architecture

Figure 5.3 represents the class diagram of the analysis and simulation tool. The class `Excecutor` can be a typical command-line or GUI based executor and is responsible to create all corresponding class instances according to data provided by the instance of class `Configuration` and provides their relation among each other. The class `Configuration` consists of the configuration data, that it parsed from the given text-based configuration file, for the current execution. The class `Logger` is a static class that cannot be instantiated and used for the logging outputs from the execution.

The interface `EventManager` provides the main definition of the functionality of *Event Manager* to be implemented by subclasses. The subclass `MMPP_EventManager` implements the MMPP for both traffic modeling and service times. The creation of `MMPP_EventManager` entails the creation of `RetransmitTimerList` and required `Distribution` class instances to obtain

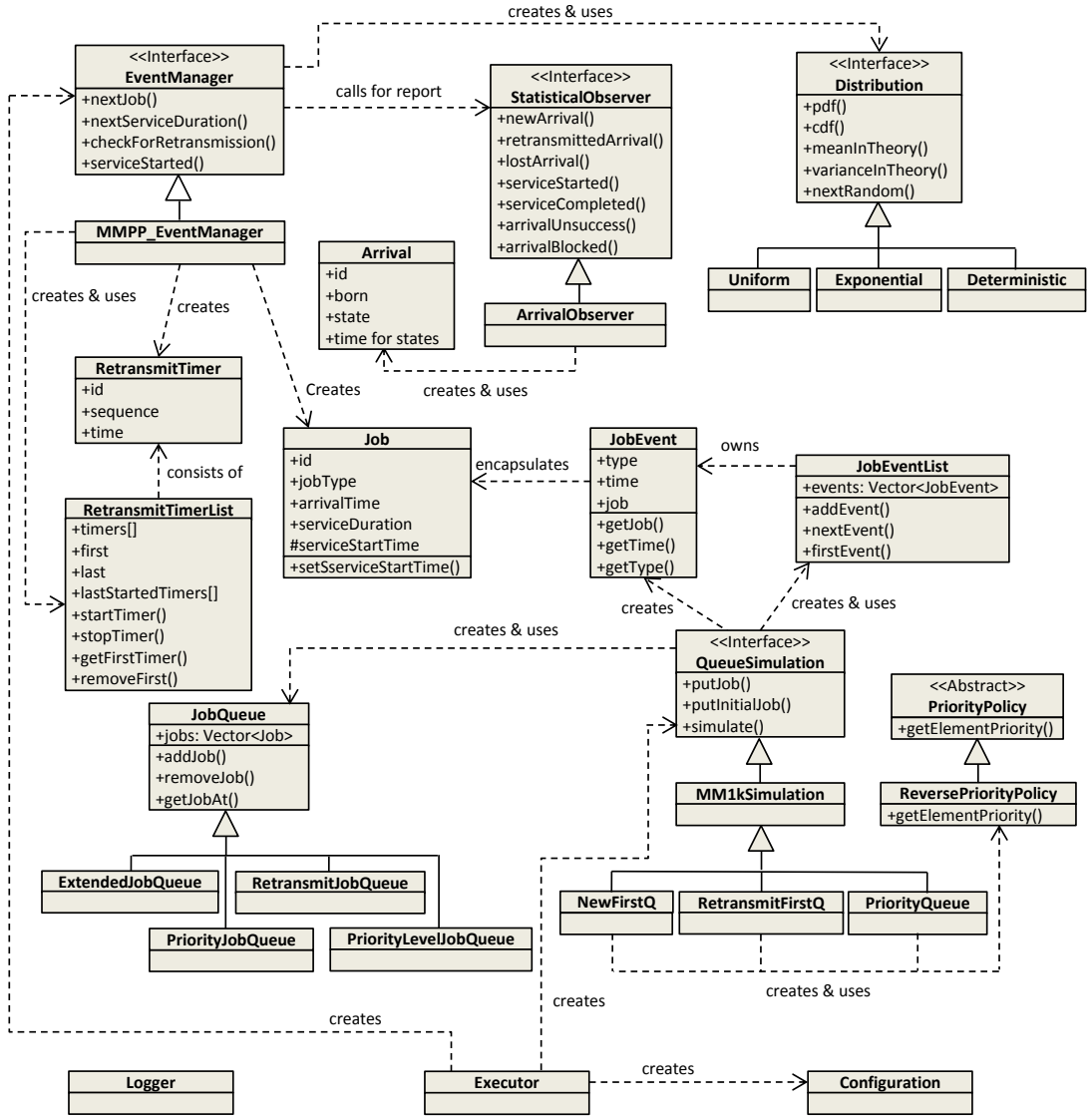


Figure 5.3 : Class diagram.

quantities for arrival time, service duration and MMPP mode parameters. According to the provisioning it can manage MMPP behavior for both arrival time and service duration quantities at the same time, as well as for one of them individually, by using MMPP mode parameters. It creates a class instance of `RetransitTimer` per arrivals and starts a timer through the class `RetransmitTimerList` by using `startTimer()` function.

The class `RetransmitTimerList` manages two types of data structures; a doubly-linked list and a hash table of the `RetransitTimer`. The doubly-linked list contains a sorted list of timer objects according to the expiry time, such that the

"head" of the list contains the first timer to expire. The hash table is used to efficient access to an instance of the `RetransitTimer`, which is based on the identifier of the corresponding arrival, to stop a timer or to find the appropriate position on the doubly-linked list for starting a timer.

The interface `QueueSimulation` provides the main definition of a queue simulation. The main implementor of this interface is the class `MM1k_Simulation` which is inspired from the Java implementation of a M/M/1 simulation in [62], which is modified to support finite buffer size behavior. It implements the conventional FIFO queue with single server, such as used for the conventional SIP server simulations. The `PriorityQueue`, the `NewFirstQ` and the `RetransFirstQ` are subclasses of it to implement non-preempted strict priority queue simulations. The class `PriorityQueue` gives higher priority either the original or retransmitted requests according to provisioned priority-policy, which is controlled by the instance of `PriorityPolicy` or `ReversePriorityPolicy`, respectively. The class `PriorityQueue` can also be configured to have a prioritization according to the sequence number of retransmissions, such as first-time retransmitted messages have higher priority over second-time retransmissions, and second-time retransmissions have have higher priority over third-time retransmissions, so on. When the `ReversePriorityPolicy` is configured for the prioritization according to the sequence number of retransmissions, sixth-time retransmissions have the highest priority, while original request messages have the lowest priority. The implementation of the class `PriorityQueue` considers that all priority classes of request messages share the same buffer size, in the case of finite buffer size. The classes `NewFirstQ` and `RetransFirstQ` to support the ability that the available buffer size of each priority class of request messages can be configured individually.

The creation of `QueueSimulation` entails the creation of the corresponding `JobQueue` according to the user specified configuration. `JobQueue` provides a data structure to queue "jobs" in the system, supporting statistical quantities such as mean queue size and mean waiting and sojourn times. It implements FIFO behavior by default. The subclass `PriorityJobQueue` is created and used by the class `PriorityQueue`, while `ExtendedJobQueue` and `RetransmitJobQueue` are used by both

NewFirstQ and RetransFirstQ. The class PriorityLevelJobQueue is created and used by PriorityJobQueue itself as an instance for each priority class of request messages.

For the queue data structures, the "jobs" are objects encapsulated into the instances of the class Job. Whenever the QueueSimulation requests an arrival from the EventManager, an instance of Job is created and encapsulated in a JobEvent to be placed into the JobEventList which schedules job-events according to their event-times, such that events can be either arrivals or departures. Then, the QueueSimulation processes job-events accordingly. Whenever a job is processed, The QueueSimulation informs the EvenetManager by calling serviceStarted() method from the interface as well as the ArrivalOserver through the interface definition StatisticalObserver by calling the method serviceStarted(). This causes the EvenetManager to stop the corresponding retransmission timer. If the removal facility is applicable for the current QueueSimulation instance, all corresponding retransmissions for this processed job are removed from the queue.

The class ArrivalObserver implements the interface defined by StatisticalObserver. The interface definition consists of methods to collect possible status information of a call, which is represented by an arrival, such as "retransmitted", "blocked", "lost", service started", "service completed" and "failed". It provides outputs for report through Logger. In the current implementation, QueueSimulation also implements the queue-observer function to collect queue size information according to PASTA (Poisson Arrivals See Time Averages) property or time intervals.

The retransmission mechanism is controlled by the EvenetManager. Because of the discrete-event driven architecture of the tool, the "current-time" of the simulation is determined by the arrival and departure times provided by the EvenetManager to the QueueSimulation. To support retransmission behavior, whenever QueueSimulation requests a new arrival by calling the method nextJob() from the interface EvenetManager, after the creating a time for a new arrival and setting that time as the "current-time", the EvenetManager checks RetransmitTimerList to verify if there are retransmit timers expired before the

"current-time". If there are, it creates a `Job` class instance for each corresponding arrival that its retransmission timer is expired, and inserts into the `JobEventList` of `QueueSimulation` by calling the method `putJob()`. A new retransmit timer is created for each retransmitted arrival according to the back-off algorithm of the SIP retransmission mechanism for the possible further retransmissions.

Lossy network behavior is handled by `EventManager`. Whenever the `QueueSimulation` requests a new arrival, the "loss-algorithm" is executed to decide whether this arrival shall be lost or not. If it is a lost arrival, it is not given to the `QueueSimulation`, and another arrival is created to pass through the algorithm until having a non-lost arrival. The `EventManager` still starts retransmission timers for lost arrivals to provide retransmitting them according to the retransmission mechanism. The same algorithm is applied retransmitted arrivals before putting them into the `JobEventList`. The algorithm is governed by a probability distribution. In the current version of the tool, we use random numbers between 0 and 1 obtained by sampling a uniform distribution (with a range between 0 and 1). For example for loss rate 10% (or for loss probability 0.1), if the value of the sample is equal to or greater than 0.1, the arrival is not considered as a loss, otherwise the arrival is considered as a loss.

5.4 Performance Tests

In this section, we conducted experiments first to test the accuracy of the analysis and simulation tool with theoretical expectations of MMPP arrival and departure models. Then, we evaluated the effect of SIP retransmission mechanism in conjunction with the result evaluated from the first set of experiments. Therefore, the first experiments are executed in the "observer" mode, while the second set of experiments are in the "applier" mode.

Our analysis and simulation tool is used to perform experiments by varying MMPP(2)/M/1/B and M/MMPP(2)/1/B parameters such as arrival and departure rates, mode durations, and buffer sizes. To create heavy traffic conditions, the ratio of overloading and non overloading mode durations is set to $1/10$ for all experiments (e.g., $\delta_0 = 1$ and $\delta_1 = 10$) as in [55]. For evaluating retransmission behavior under various queuing dynamics, ten different MMPP mode duration parameters starting

from $\delta_0(0) = 1$, $\delta_1(0) = 10$) to ($\delta_0(9) = 0.00195$, $\delta_1(9) = 0.01953$) are used. Note that the mode duration values are obtained by iteratively taking half of the previous values such as ($\delta_0(1) = \delta_0(0)/2 = 0.5$, $\delta_1(1) = \delta_1(0)/2 = 5$).

To evaluate the effects of the buffer size of the SIP server, B is varied from 100 to 1000 with a step size of 100 so that ten different B values are used. A simulation scenario is defined by setting MMPP mode parameters (δ_0 and δ_1), B, and MMPP queue model type together with its arrival and departure rates. The simulation time for each scenario is set to 6000 seconds. The simulation for each scenario is repeated 100 times and results are reported as average values of 100 experiments with a 95% confidence interval. Numerical results are reported for two running modes: *(i)* retransmission mechanism is disabled, in which no retransmission messages are sent but only retransmission conditions are observed and reported (*observer mode*), *(ii)* retransmission mode is enabled (*applier mode*), in which retransmission messages are sent when the corresponding timer expire. For all experiments, the SIP server silently drops requests when the buffer is overflowed, except the buffer size is set to pseudo-infinite. We assume that the SIP server does not apply any specific overload control mechanism.

5.4.1 Test results for observer mode

Observer mode experiments consist of tests for MMPP(2)/M/1/B and M/MMPP(2)/1/B queues, respectively.

5.4.1.1 Results for MMPP(2)/M/1/B

The mean arrival rate for bursty traffic mode is λ_0=995 messages/sec, the mean arrival rate for normal traffic is λ_1=199 messages/sec, and the mean service rate is μ=1000 messages/sec. The average arrival rate can be calculated with Eq.(5.1).

Since the ratio of mode duration values selected in this work is constant (i.e., $\delta_0(i)/\delta_1(i) = 1/10$, $i = 0, 1, .., 9$); the theoretical average arrival rates are calculated as 922.6363 messages/sec for all of ten cases. The reported average arrival rates are within 95% confidence interval of this theoretical value. We also observed that smaller values of mode durations provide higher mean queue sizes which increase the overloading probability. We calculated the theoretical mean queue size and blocking

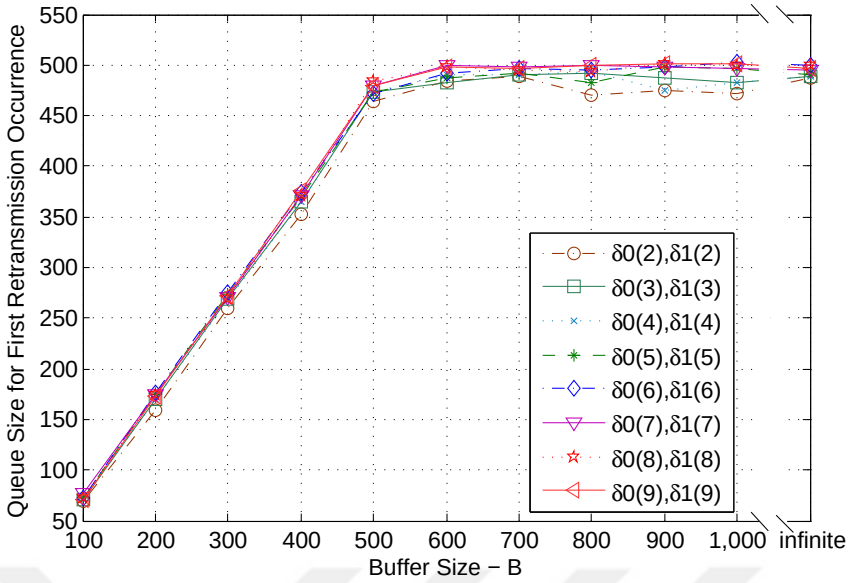


Figure 5.4 : Queue sizes that first retransmission occurs for MMPP(2)/M/1/B.

probabilities by using state probabilities calculated with the software program in [60]. Table 5.1 compares the theoretical expectations with results obtained from the tool, for MMPP(2)/M/1/600 with mode durations ($\delta_0(9) = 0.00195$, $\delta_1(9) = 0.01953$). For the theoretical calculation, error criteria is selected as 10^{-6}, and the program is ended after 124,839 iterations.

Table 5.1 : Comparison between theoretical evaluates and tool results for MMPP(2)/M/1/600.

	Mean Queue Size	**Blocking Probability**
Theoretical	147.39	2.30e-4
Tool	149.16	2.33e-4

To record a retransmission, either the buffer size should be full or the end-to-end delay should be higher than a certain threshold value [9, 19]. Note that, in this test case, only the queuing delay is considered while the processing, transmission, and propagation delays are ignored. The queuing delay greater than T_1 (i.e., $q/\mu \geq T_1$) triggers the retransmission event for a SIP request [55]. In other words, when the queue size (q) of a SIP server exceeds $\mu \times T_1$ messages, the first retransmission for the corresponding request is sent. For T_1=0.5 sec and μ=1000 messages/sec, the first retransmission occurrence is expected when q is around 500 for the cases of $B \geq 500$. This corresponds to the queuing delay of about 0.5 seconds, which likely triggers the first retransmission event. Figure 5.4 illustrates average queue sizes that first retransmission condition is

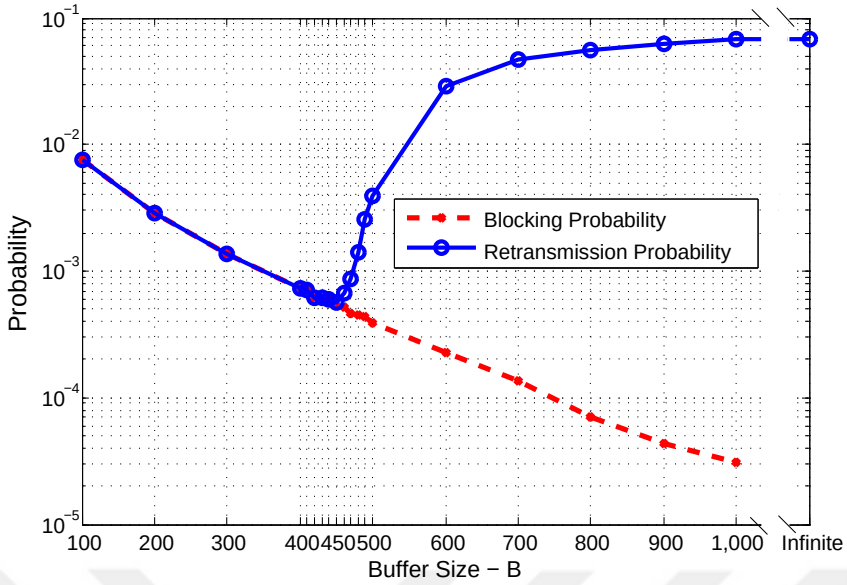


Figure 5.5 : Blocking and retransmission probabilities for MMPP(2)/M/1/B.

recorded with respect to B. Experimental results verified theoretical expectations for all mode durations.

The retransmission probability can be calculated as follows:

$$P_{rt} = P(q_{i0} \geq \mu T_1) + P(q_{i0} \geq \mu T_1) + P_b \tag{5.2}$$

where P_b denotes the blocking probability, q_{i0} and q_{i1} denote queue sizes for overloading and non-overloading mode durations, respectively ($0 \leq i \leq B$). Figure 5.5 illustrates P_{rt} and P_b with respect to B for ($\delta_0(9) = 0.00195, \delta_1(9) = 0.01953$). Additional experiments are performed for buffers sizes between 400 and 500 with a step size of 10 to observe the behavior of P_{rt} and P_b in higher resolution. As shown in the figure, P_{rt} is close to P_b when $B < 450$ whereas P_{rt} starts splitting off from P_b when $B > 450$. In this case of $B < 450$, the retransmission can occur solely due to a blocking event. P_{rt} becomes higher than P_b when $q \geq 450$ since an additional queuing delay becomes more effective at these regimes. P_{rt} should equal to P_b according to Eq. 5.2 when $B < 500$, simulation experiments showed that this is true for $B < 450$.

5.4.1.2 Results for M/MMPP(2)/1/B

In this section, the mean arrival rate is $\lambda = 199$ messages/sec, the mean service rate at maintenance state is $\mu_0 = 200$ messages/sec, and the mean service rate at the normal service state is $\mu_1 = 1000$ messages/sec. These values are obtained from [55] to have

a reference model for comparison and selected to provide the same utilization level as the MMPP(2)/M/1/B arrival and departure parameters used in the previous section. Similar to MMPP(2)/M/1/B, numerical experiments showed that smaller values of mode durations provide higher mean queue sizes in the M/MMPP(2)/1/B case as well. Table 5.2 compares the theoretical expectations with results obtained from the tool, for M/MMPP(2)/1/600 with mode durations ($\delta_0(9) = 0.00195$, $\delta_1(9) = 0.01953$). State probabilities calculated with the software program in [61]. For the theoretical calculation, error criteria is selected as 10^{-6}, and the program is ended after 131,634 iterations.

Table 5.2 : Comparison between theoretical evaluates and tool results for M/MMPP(2)/1/600.

	Mean Queue Size	**Blocking Probability**
Theoretical	130.24	1.46e-4
Tool	126.97	1.37e-4

Note that Eq. 5.2 is applicable also for M/MMPP(2)/1/B model, but we have two mean service rates. In this case, by using the default value 0.5 sec for T_1, retransmission probability can be calculated as:

$$P_{rt} = P(q_{i0} \geq 100) + P(q_{i0} \geq 500) + P_b \tag{5.3}$$

Note that retransmission events likely occur when the queue size is around 100 according to the M/MMPP(2)/1/B model parameters. Figure 5.6 verifies that first retransmissions are triggered when the server's queue size is around 100. Figure 5.7 illustrates P_{rt} and P_b with respect to B for ($\delta_0(9) = 0.00195$, $\delta_1(9) = 0.01953$) for M/MMPP(2)/1/B with in higher resolution between buffer size values 10 and 100. P_{rt} starts splitting off from P_b about buffer size of $B = 80$, and it takes higher values than the blocking probability for $B \geq 100$.

5.4.2 Test results for applier mode

Unless otherwise stated, for all experiments in this section, the same MMPP parameters (arrival and departure rates and mode durations) used in the observer mode, are used in the applier mode; however, the applier mode enables the retransmission mechanism. Since INVITE requests can be retransmitted in addition to the original requests, the underlying queuing type in the applier mode can not be modeled using

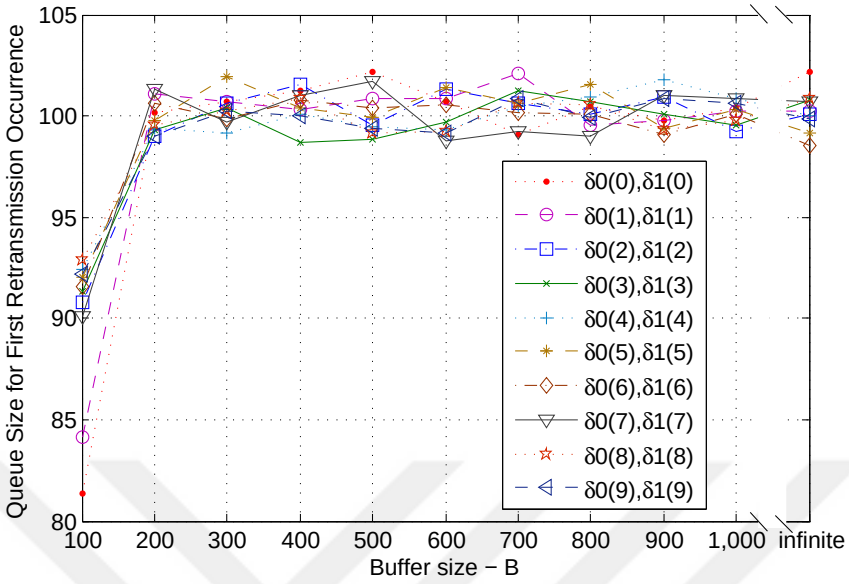


Figure 5.6 : Queue sizes that first retransmission occurs for M/MMPP(2)/1/B.

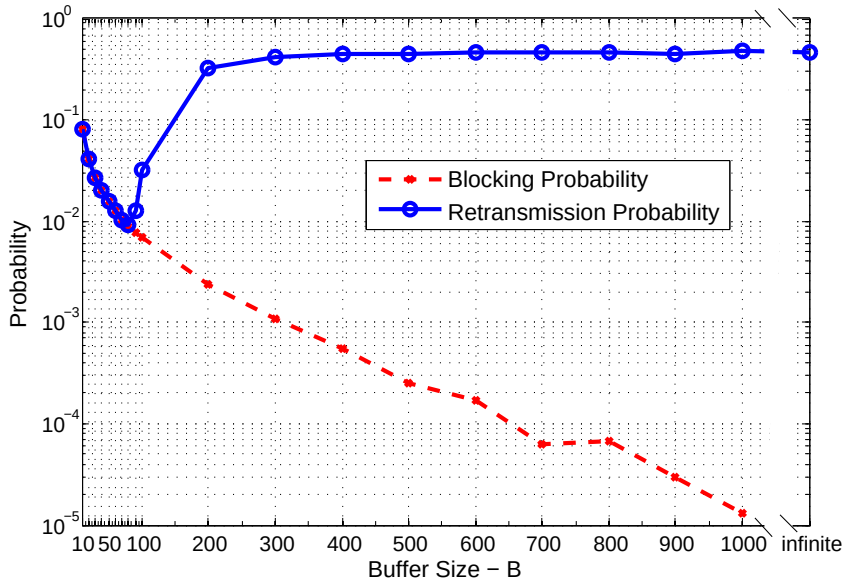


Figure 5.7 : Blocking and retransmission probabilities for M/MMPP(2)/1/B.

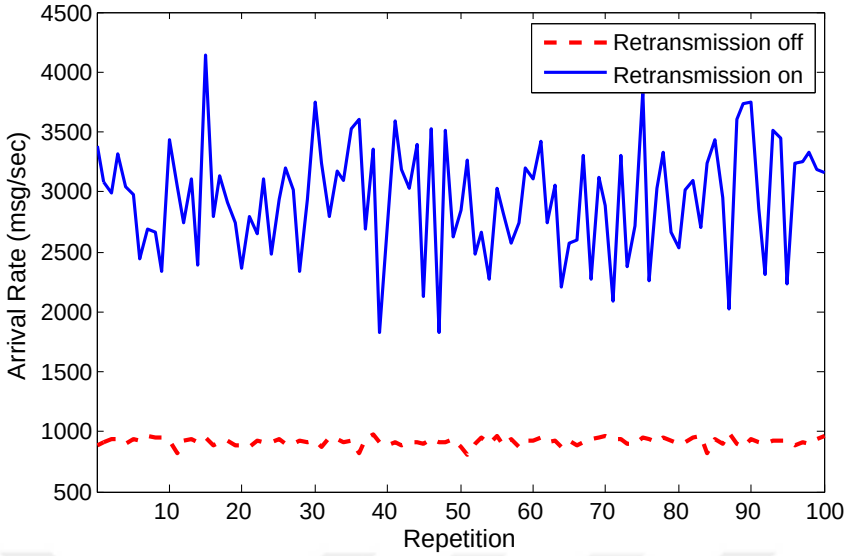


Figure 5.8 : The effects of the retransmissions on the arrival rate for MMPP(2)/M/1/1000.

MMPP(2)/M/1/N or M/MMPP(2)/1/N anymore. However, using the same MMPP parameters will allow us to identify the effects of the retransmission on the SIP server overloading by using the results of the observer mode as a baseline. For all experiments, without loss of generality, we observed that the total arrival rates to the SIP server significantly increases when retransmission events occurred. The total arrival rates to the SIP server in terms of messages/sec are shown in Figure 5.8 and Figure 5.9 for MMPP(2)/M/1/1000 and M/MMPP(2)/1/1000, respectively. The MMPP mode durations are set to $(\delta_0(9) = 0.00195, \delta_1(9) = 0.01953)$ and 100 random experiments are performed. Test results showed that, the total arrival rates increased about 3 and 5 times in the applier mode compared to the observer mode for MMPP(2)/M/1/1000 and M/MMPP(2)/1/1000, respectively. The adverse effects of the increased arrival rates on the queuing dynamics are depicted in Table 5.3 for MMPP(2)/M/1/1000 and M/MMPP(2)/1/1000. Test results show that the mean queue size ($\bar{q}$), blocking and retransmission probabilities are significantly higher in the applier mode.

If retransmissions are triggered because the server has already been overloaded, additional retransmission traffic makes the server overload worse. For example, for the experiments using a sufficiently large buffer size (simulating infinite buffer) and

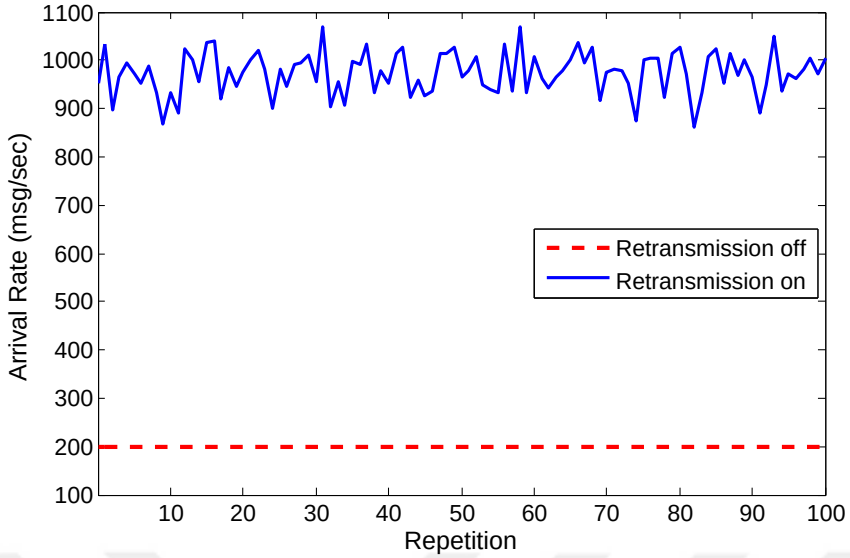


Figure 5.9 : The effects of the retransmissions on the arrival rate for M/MMPP(2)/1/1000.

Table 5.3 : Observed effects of retransmission at queue model with B=1000 and for mode parameters $\delta_0(9) = 0.001953$ and $\delta_1(9) = 0.019531$.

	MMPP(2)/M/1/1000			**M/MMPP(2)/1/1000**		
	$\bar{q}$	P_b	P_{rt}	$\bar{q}$	P_b	P_{rt}
Observer M.	163.53	4.373e-5	0.0625	138.75	2.950e-5	0.4755
Applier M.	614.70	0.6653	0.6711	867.12	0.7860	0.7951

lower mode durations (e.g., for $\delta_i(9), \delta_i(8), \ldots, \delta_i(4)$, where $i = 0, 1$), the simulations were crashed due to out-of-memory problem (the queue size keeps increasing with excessive retransmissions).

Figure 5.10 illustrates the ratio of the number of retransmissions to the number of the original SIP requests with respect to B for MMPP(2)/M/1/N. The mode durations are set to ($\delta_0(9) = 0.00195, \delta_1(9) = 0.01953$) and B is varied from 100 to 1000, with a step size of 100 (fine granular experiments between 400 and 500 with a step size of 10). The ratios are given for each retransmission sequence such that *1st retrans* in the legend represents the ratio of the number of the first retransmissions to the number of the original requests. *retrans. ceases* represents the ratio of the number of unsuccessful SIP calls to the original SIP calls according to the transaction timeout based on T_B timer of SIP.

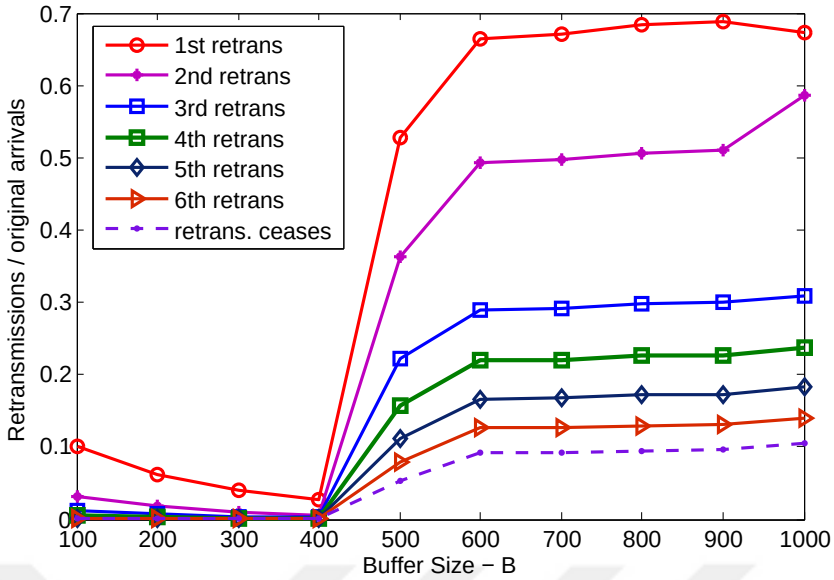


Figure 5.10 : The ratio of the number of retransmissions to the number of the original arrivals for MMPP(2)/M/1/B.

Test results show that when B is higher than 400, the number of retransmissions gets higher as B increases and the rate of the change significantly decreases after B=600. When B is smaller than 400, the number of retransmissions gets higher as B decreases. The main reason for this behavior is that there are two dynamics which affect the number of retransmissions; (i) the blocking probability, and (ii) the queuing delay. When B is higher than 400, the queuing delay becomes a dominating factor since allowing higher queue sizes yield higher queuing delay and hence more likely retransmissions. When B is smaller than 400, the blocking probability becomes a dominating factor since the buffer size of less than 400 never results in a queuing delay higher than 0.5 sec which is a required delay for the first retransmission. Smaller B values yield higher blocking probability; and hence higher number of retransmissions. Test results indicate that the number of total retransmissions will significantly decrease if an incoming request is dropped instead of buffered when the queuing delay of the server has an approximate value of the first retransmission timer. Although the first retransmissions are triggered by blocking an incoming request, we provide enough buffer spaces for the subsequent requests. The same pattern is observed for all experiments when the MMPP(2)/M/1/B parameters are used in the applier mode.

Similarly, as shown in Figure 5.11, for the M/MMPP(2)/1/B parameters used in the applier mode, the number of retransmissions is minimized when B is around 100

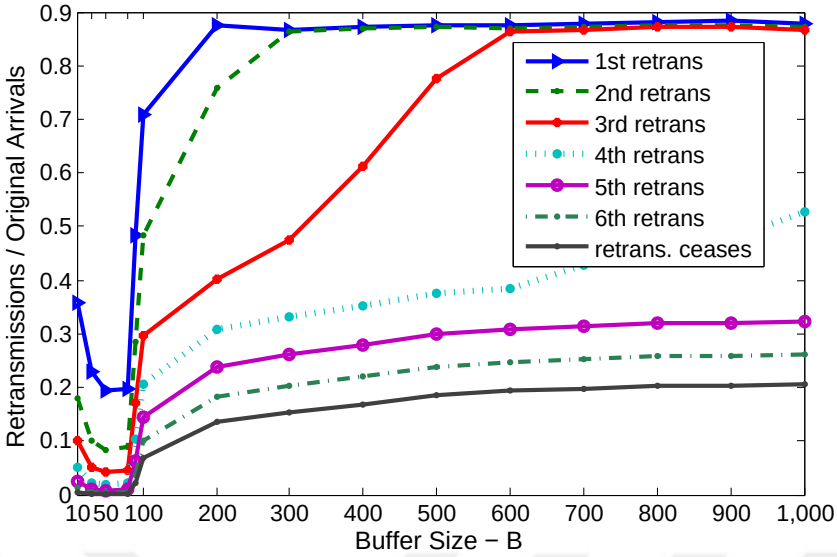


Figure 5.11 : The ratio of the number of retransmissions to the number of the original arrivals for M/MMPP(2)/1/B.

because of the above dynamics. This is because of the queuing delay getting about 0.5 sec when B is around 100.

5.5 Tests for PRSM

Our aim in this section is to evaluate the performance of the PRSM by using the analysis and simulation tool presented in this chapter. We compare the performance of two schemes: *(i)* The conventional SIP, and *(ii)* SIP server with the PRSM. We selected 2-mode MMPP parameters as $\delta_0 = 0.00195$ and $\delta_1 = 0.01953$ to emulate heavy traffic conditions. The mean arrival rate for the heavy traffic mode is set to $\lambda_0 = 995$ messages/sec; the mean arrival rate for the light traffic mode is set to $\lambda_1 = 199$ messages/sec; the mean service rate is set to $\mu = 1000$ messages/sec. Note that the mean service rate corresponds to the completion times of message parsing and retransmission detection processes as defined in RFC 3261 up to sending 100-Trying response message. The time required for pre-parsing and removing retransmissions is assumed to be negligible compared to the message parsing and retransmission detection. The numerical experiments performed using JAIN-SIP stack verifies this assumption (see Chapter 7 for details).

The ratio of the number of retransmissions to the number of total arrivals has been defined as the performance evaluation measure. To evaluate the effects of the buffer

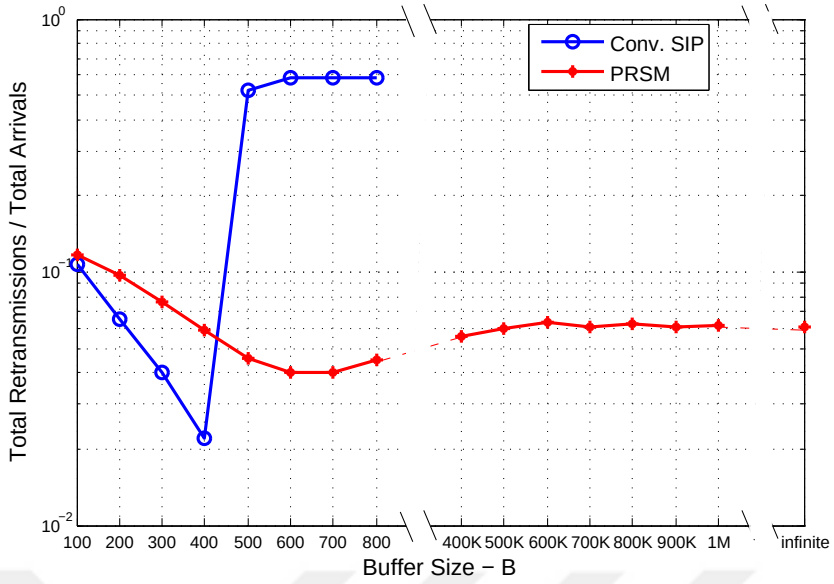


Figure 5.12 : The ratio of retransmissions to total arrivals versus B.

size on the performance of PRSM, B is varied from 100 to 1M with non-regular step sizes. The simulation time for each scenario is set to 6000 seconds. The results are reported as the average values of 100 experiments with a 95% confidence interval.

The ratio of the number of retransmissions to the number of total arrivals for both the conventional SIP and the PRSM are plotted in Figure 5.12. To observe a retransmission event, either the buffer size should be full or the end-to-end delay should be higher than a certain threshold value. In our model, we considered only queuing delay, and ignored transmission and propagation delays. Hence, the queuing delay greater than T_1 (i.e., $q/\mu \geq T_1$) triggers the retransmission event for a SIP request. In other words, when the queue size (q) of a SIP server exceeds $\mu \times T_1$ messages, the first retransmission for the corresponding request is sent. According to Eq.(5.2), for $T_1 = 0.5$ sec and $\mu = 1000$ messages/sec, the first retransmission occurrence is expected when q is around 500 for the cases of $B \geq 500$. This corresponds to the queuing delay of about 0.5 sec, which likely triggers the first retransmission event. According to SIP server scheduling, there are two dynamics which affect the number of retransmissions; *(i)* the blocking probability, and *(ii)* the queuing delay. When B is smaller than 500, blocking probability is effective on retransmissions. As shown in Figure 5.12, higher B values causes additional retransmissions in the conventional SIP mechanism because of higher queuing delay and hence confirms theoretical expectations.

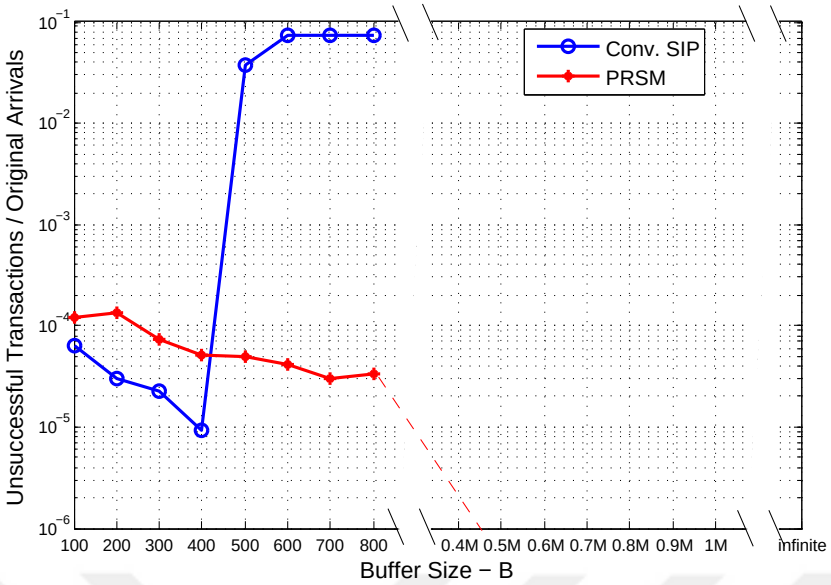


Figure 5.13 : The ratio of retransmissions to total arrivals versus B.

When B is equal to or higher than 500, the PRSM yields lower number of retransmissions compared to the conventional SIP since responding new arrivals first during the server overloading limits the number of the first retransmissions. For smaller B values, retransmissions are likely non-redundant since higher blocking probability causes new arrivals being dropped from the buffer before the server process them. Processing retransmissions of blocked requests in lower priority for the PRSM triggers consecutive retransmissions and hence the PRSM performs worse than the conventional SIP when B is lower than 500. When B is higher than 500, the PRSM reduces the ratio of the total retransmissions to total arrivals to around 0.06.

Note that, when the conventional SIP without the PRSM capability is used at high buffer sizes (>10K), the tool crashes due to the excessive number of retransmissions yielding high queue occupancies. The detailed analyses indicate that the conventional SIP crashed for the finite buffer sizes beyond 10K due to the out of memory error. Figure 5.12, Figure 5.13 and Figure 5.14 show that when the conventional SIP with the PRSM capability is used, we never observed a server crash. For the buffer size values up to 800, the PRSM yields the unsuccessful transaction ratio between $1.19x10^{-4}$ and $3.31x10^{-5}$. Note that the ratio of the unsuccessful transactions to the original arrivals in Figure 5.13 is around 10^{-6} when B is 1000 and becomes zero when B is beyond 100K. Figure 5.14 illustrates the mean waiting time for 100-Trying for various buffer sizes to the original arrivals. The behavior of these performance metrics is similar to

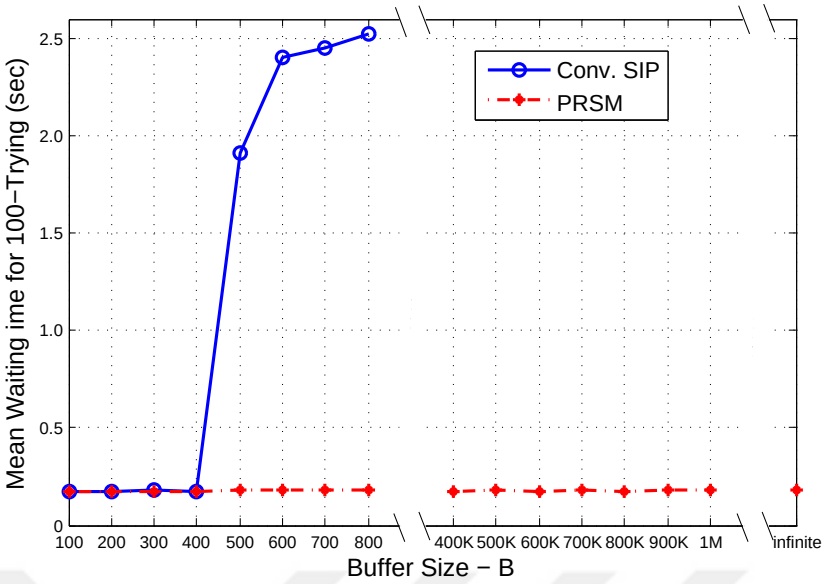


Figure 5.14 : Mean waiting time for 100-Trying response versus B.

the behavior of the retransmission ratio. The response delay for the PRSM is about 180 msec for all buffer sizes. When B is set to a pseudo-infinite value, the PRSM responds all requests successfully within 64 $\times T_1$ seconds.

5.6 Test for Lossy Network Conditions

Simulation of the message loss is added to the tool to support experiments under lossy network conditions. Although our initial implementation considers only loss rates from the uniform distribution, the tool can be easily extended to add other types of distributions. We assume that the loss behavior is not applied for response messages, but only original and retransmitted request messages. We conducted experiments to evaluate the behavior of SIP retransmission mechanism under lossy network conditions for both the conventional SIP and the PRSM.

We choose the test case from [8], which is already used to investigate the stability condition of overloaded SIP servers having infinite buffer. In this test case, the server is overloaded by initial bulk arrivals arrived at $t = 0s$, which are queued in the server to create an initial queue size, to emulate a short surge of use demands. After a demand burst, the original request message arrives at the overloaded server with a constant rate $\lambda = 200$ messages/sec, emulating a regular user demand. The overloaded server maintains a constant service rate $\mu = 1000$ messages/sec. Thus, the effective utilization

Table 5.4 : Number of lost messages compared with the number of arrived messages for 10% loss rate.

	Number of Received Msg.	**Number of Lost Msg.**
Original Messages	30000	2989.89 ± 10.32
1st Retransmissions	26603.99 ± 31.54	2667.89 ± 7.56
2nd Retransmissions	24340.11 ± 34.18	2436.39 ± 9.1
3rd Retransmissions	21115.35 ± 34.14	2109.64 ± 8.31
4th Retransmissions	14701.01 ± 34.28	1471.59 ± 6.45
5th Retransmissions	31.55 ± 1.05	2.92 ± 0.36
6th Retransmissions	0.0099 ± 0.0196	0

for the regular user demand is $\rho = \lambda/\mu = 20\%$. Both arrival and service rates are deterministic. We set the amount of the initial bulk arrivals as 6000 which violets the stability conditions of the server according to the given test parameters. We extended the simulation period to be 150 seconds. We used various loss rates between 0%-90% with steps of 10% to be able to evaluate the effects of high network losses, and for the sake of clear presentation we only report some of results. We assume that initial bulk arrivals arrived the server without any loss in the network, therefore expected retransmissions for these arrivals are not because of the loss messages, but because of processing delay in the server. Besides, corresponding retransmissions of these initial bulk arrivals because of processing delay may get lost in the network.

First, we demonstrate the loss capability of the tool by reporting the amount of the lost messages of both original and retransmissions under various target loss rates. Although arrival and service rates are deterministic, because of the randomness of the loss rate, we execute the test case 100 times and the results are reported as the average values of 100 experiments with a 95% confidence interval. Table 5.4 gives the results for 10% loss rate. As shown in the table, the tool can yield accurate loss rates for the simulation.

Figure 5.15 and Figure 5.16 illustrate queue size and retransmission rate behavior versus time for the conventional SIP server with various loss rates as well as no-loss cases. For the detailed analysis of the behavior of the queue size and retransmission rate, we refer to [8] and Section 6.1.1 of this thesis under the condition that there is no loss in the network. From figures, in general, it can be observed that lossy network conditions decrease the amount of the offered traffic to the SIP server. Therefore, for the lossless case, while the queue size of the conventional SIP server increases almost linearly (refer to Figure 5.15) because of the aggregated retransmission traffic reaching

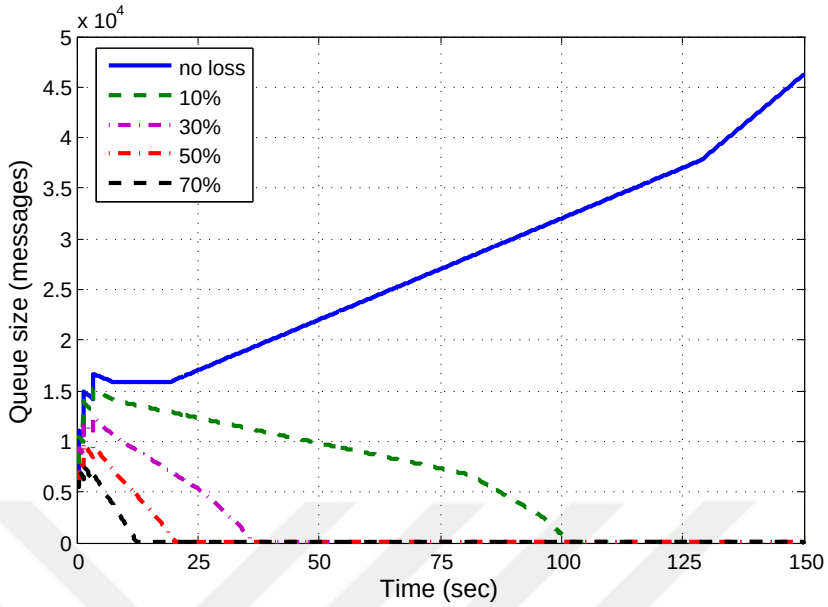


Figure 5.15 : Queue size behavior of the conventional SIP server with various loss rates.

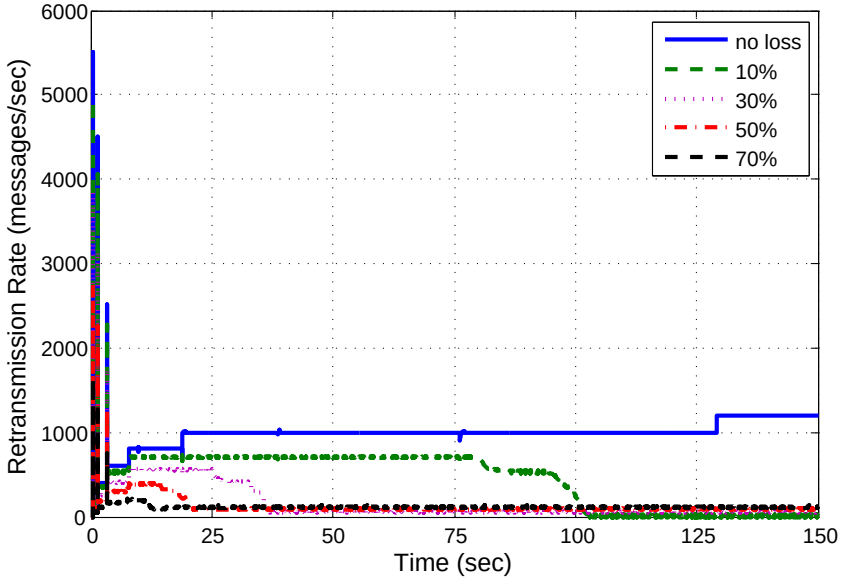


Figure 5.16 : Retransmission rate behavior of the conventional SIP server with various loss rates.

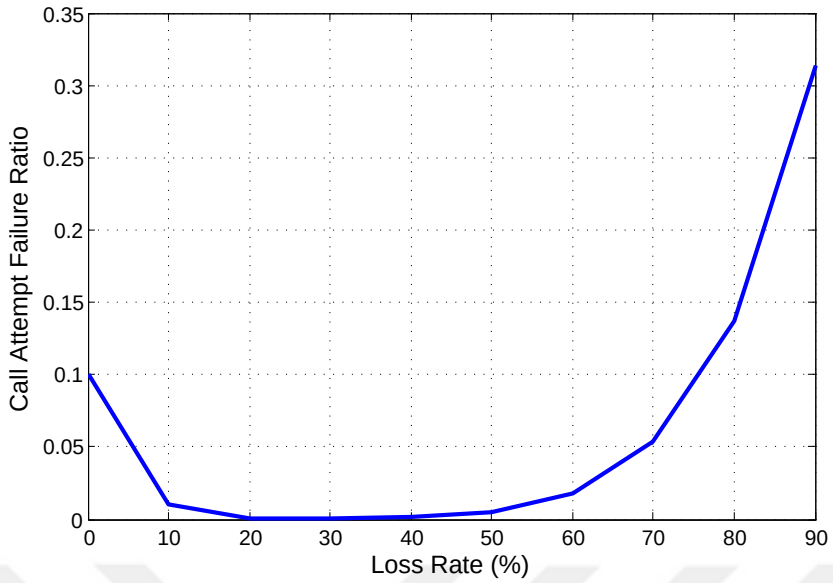


Figure 5.17 : Call failure ratio for the conventional SIP with various loss rates.

1000 messages/sec at almost t=19 seconds, the queue size decreased to almost zero at t=100 seconds even for the loss rate 10%. This is due to the fact that the retransmission rates do not reach higher levels as opposed to the case that there is no loss in the network. This shows that the conventional SIP server can find enough power to process reduced original arrival and retransmission rates. Besides, the failed call attempt ratio is decreased as a positive effect of the retransmission mechanism for loss rates up to 70%, when compared with the no-loss case (refer to Figure 5.17).

Figure 5.18 shows that the average delay of the conventional SIP server is also decreased as the effect of reduced offered traffic. Figure 5.19 and Figure 5.20 report delays of each individual call in the order of response and the running average [63] (recursive averaging [64]) delay for no-loss and 10% loss rate cases, respectively. The singular peaks indicate the processing delay of retransmitted messages of calls that original message and earlier retransmissions are lost in the network. These results are obtained from a single execution of the test case. Note that, in the case of loss rate 10%, the conventional SIP server can process 35,989 of 36,000 calls (6000 + 200 $\times$ 150), while 29,001 in the case of no-loss in the network.

When we applied the test case for the PRSM, for lossless network condition, the total queue size and retransmission rate are not affected by the bulk arrivals violating the stability condition of the conventional SIP server, as shown in Figure 5.21 and

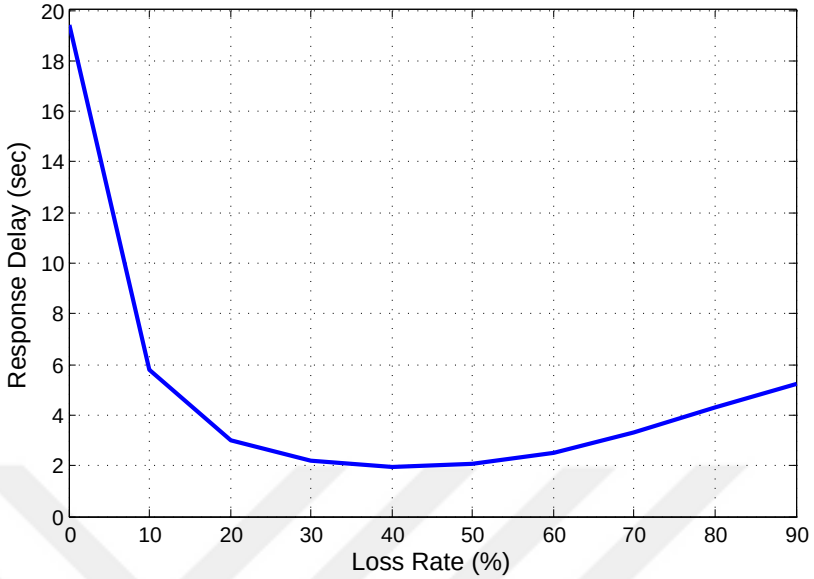


Figure 5.18 : Average delay for the conventional SIP according to various loss rates.

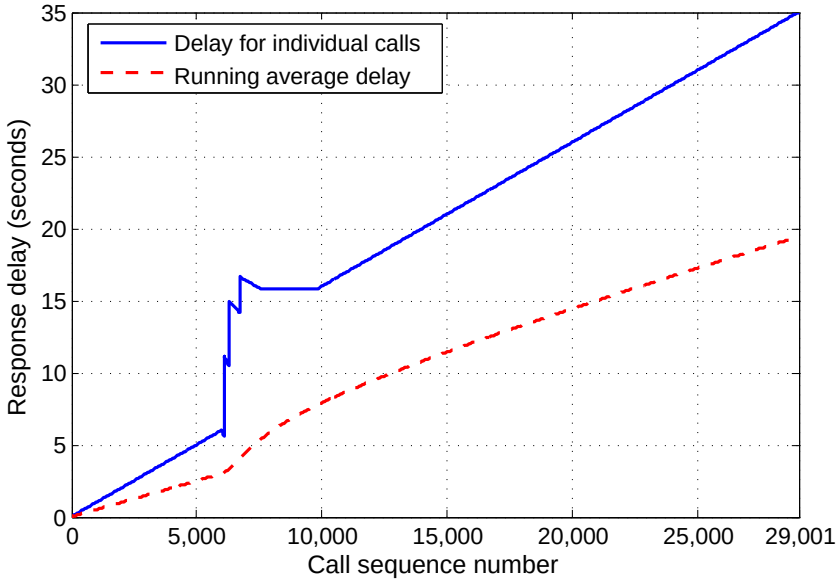


Figure 5.19 : Delay for individual calls in sequence and the running average delay for the conventional SIP for no-loss case.

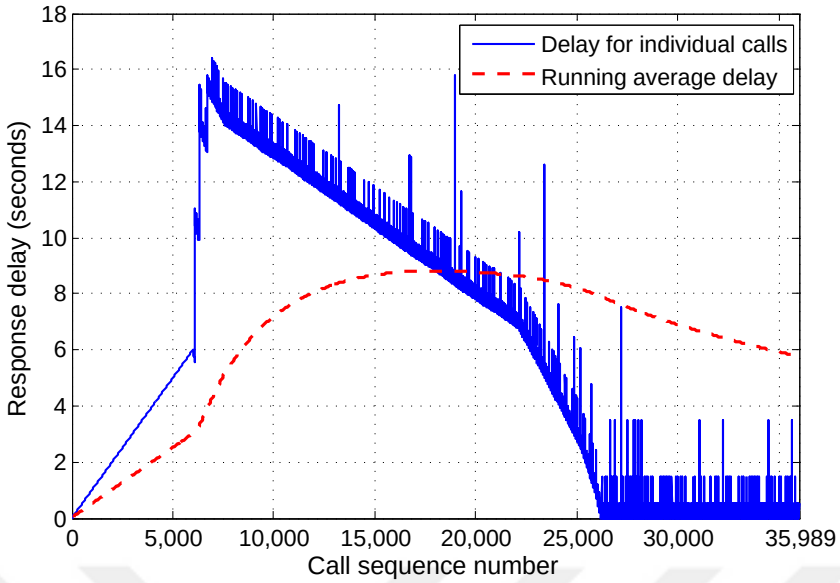


Figure 5.20 : Delay for individual calls in sequence and the running average delay for the conventional SIP for loss rate 10%.

Figure 5.22, respectively. This is because of preferring to process original arrivals first and removing the corresponding redundant retransmissions (refer to Section 6.1.1 for details). Besides, for lossy network conditions, because of reduction in the offered load for the server, the queue size and retransmission rate are decreased correspondingly. Figure 5.23 and Figure 5.24 depict call failure ratio and average response delay for loss rates from 0% to 90%. Both performance measures behave similar to the conventional SIP case for higher loss rates.

Figure 5.25 and Figure 5.26 show delays of each individual call in the order of response and the running average (recursive averaging))delay for no-loss case and for loss rate 10%, respectively. The singular peaks indicate the processing delay of retransmitted messages corresponding to the calls that their original message and earlier retransmissions are lost in the network. Each result is obtained from a single execution of the test case. When the loss rate gets higher, the response delays for retransmissions are increased because of increase in the amount of non-redundant retransmissions. Figure 5.27 illustrates delays per call for loss rate 20% and 30%, respectively.

When we disable the removal process of the PRSM, the queue size behavior is different compared to the PRSM with the removal process as shown in Figure 5.28, however, the

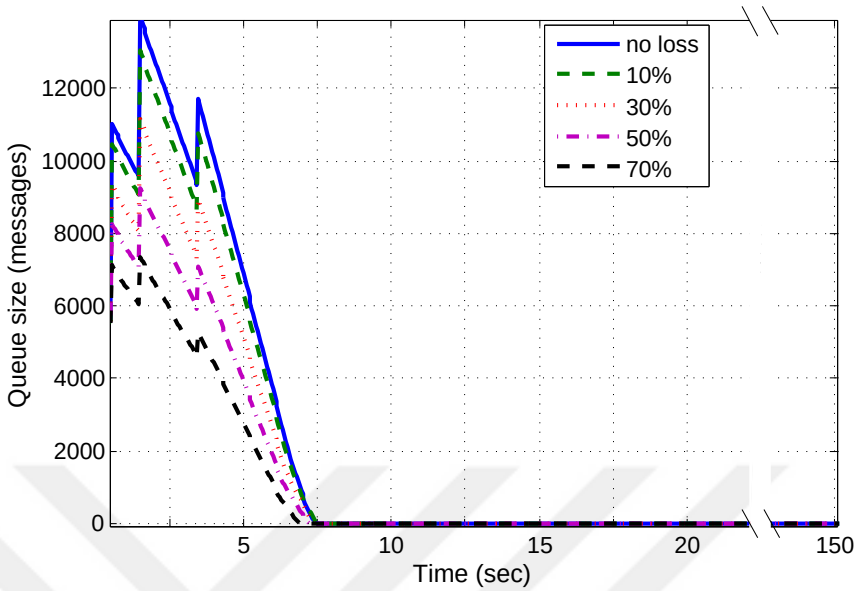


Figure 5.21 : Total queue size behavior of the PRSM with various loss rates.

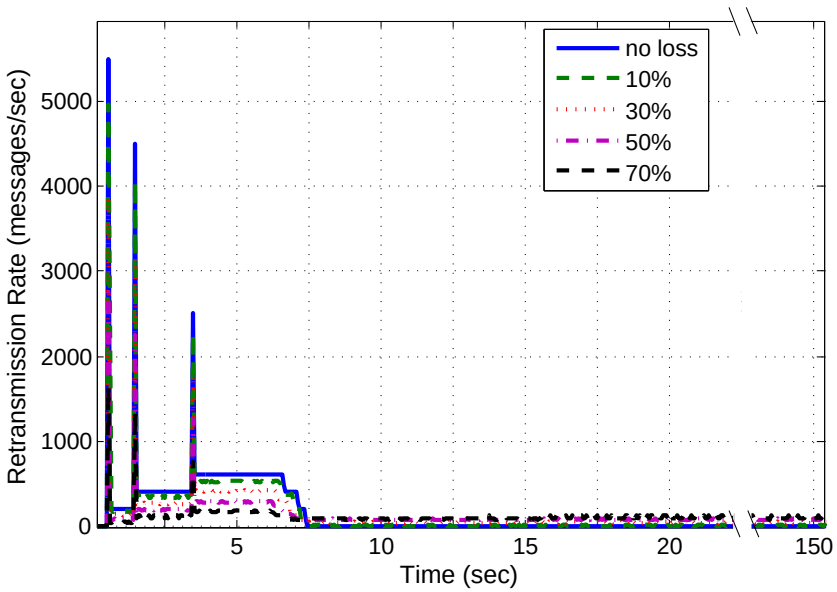


Figure 5.22 : Retransmission rate behavior of the PRSM with various loss rates.

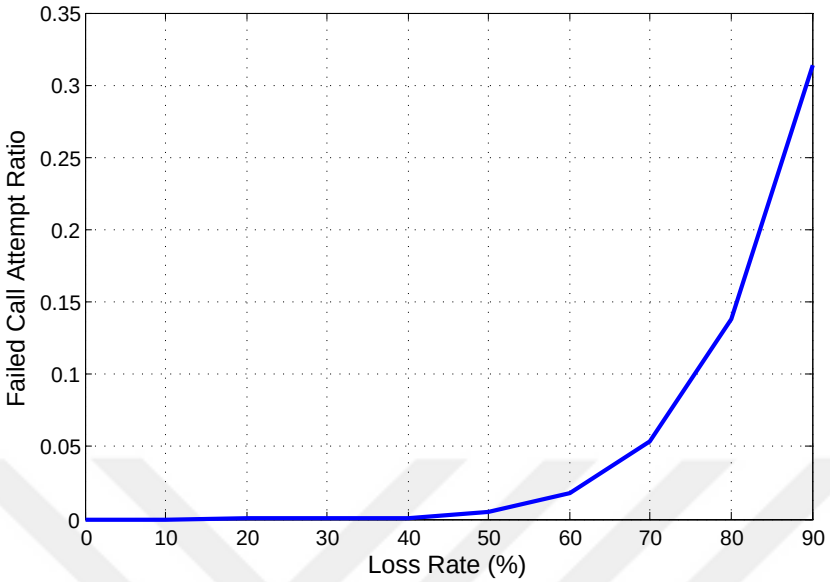


Figure 5.23 : Call failure ratio for the PRSM according to various loss rates.

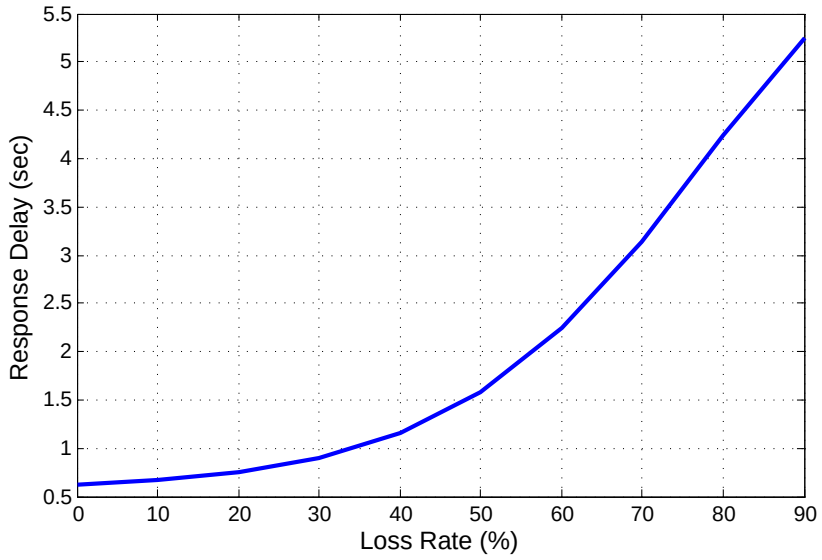


Figure 5.24 : Average delay for the PRSM according to various loss rates.

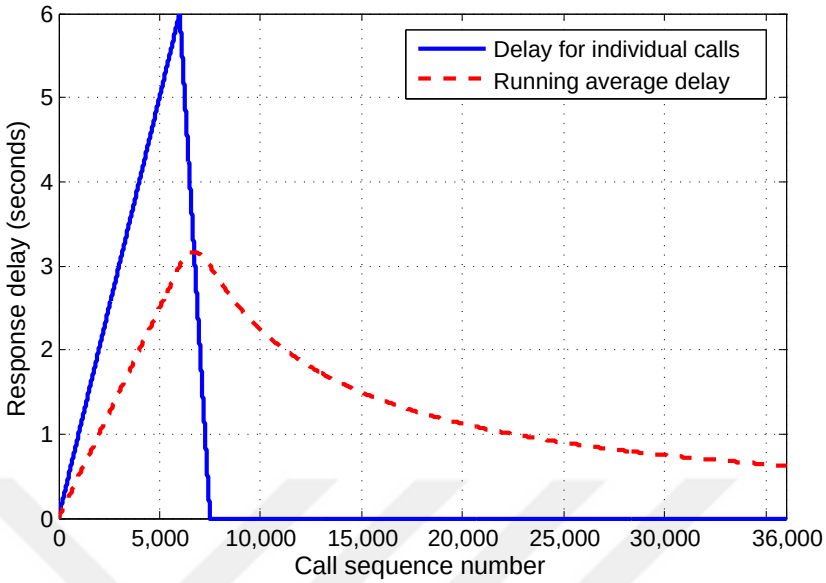


Figure 5.25 : Delay for individual calls in sequence and the running average delay for the PRSM for no-loss case.

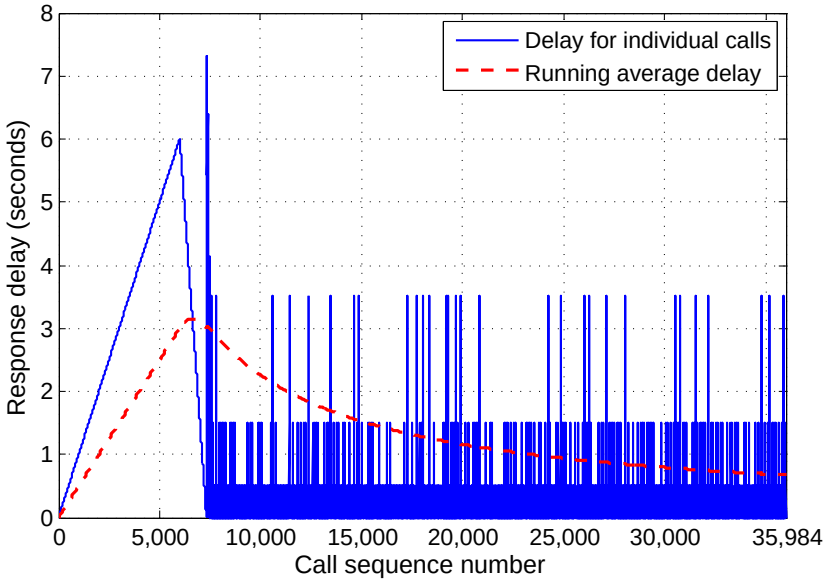
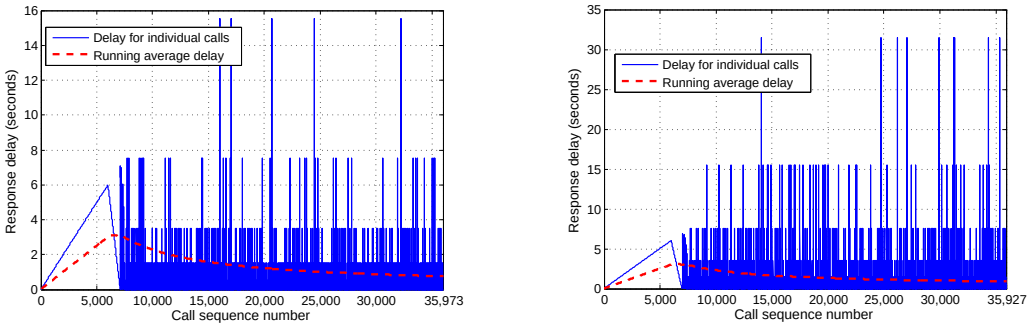


Figure 5.26 : Delay for individual calls in sequence and the running average delay for the PRSM for loss rate 10%.



(a) (b)

Figure 5.27 : Delay for individual calls in sequence and the running average delay for the PRSM for loss rates (a) 20% and (b) 30%, respectively.

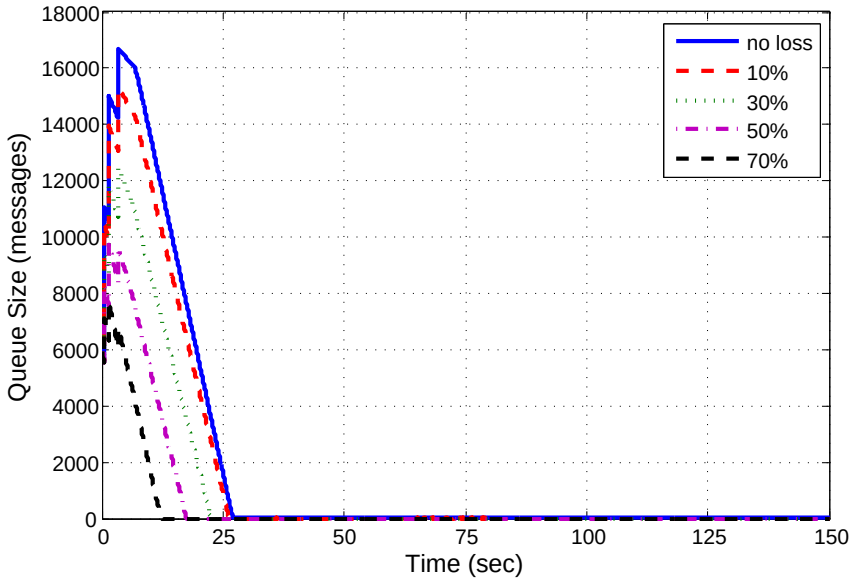
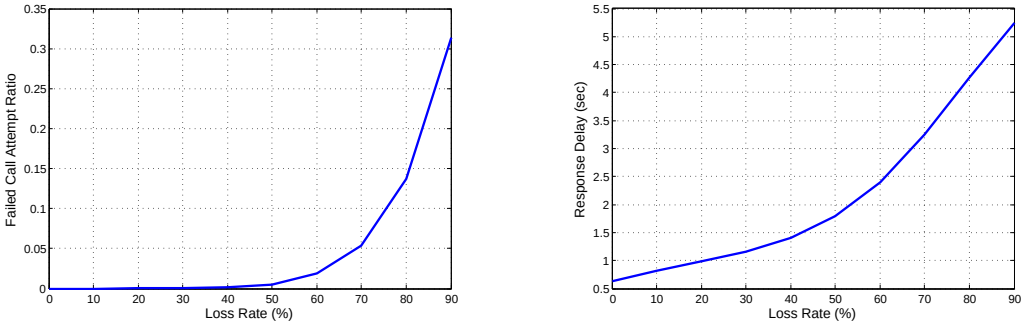


Figure 5.28 : Total queue size behavior of the PRSM with removal deactivated for various loss rates.



(a) (b)

Figure 5.29 : (a) Failed call attempt ratio for the PRSM with no removal according to various loss rates, (b) Average delay for the PRSM with no removal according to various loss rates.

retransmissions rate behavior is the same as shown in Figure 5.22. The reason of these results is that retransmitted messages are processed by the server when the primary queue is empty. Since the server gives the highest priority to the original messages, it limits the first retransmissions and hence the number of subsequent retransmissions in the same way as the PRSM with the removal process. But, because of there is no removal of redundant retransmissions, the server has to process also redundant retransmissions form the queue. As a result of this, the processing of the non-redundant retransmissions will be highly delayed when compared to the PRSM with removal as shown in Figure 5.29. Figure 5.30 shows delays per call for loss rate 10%.

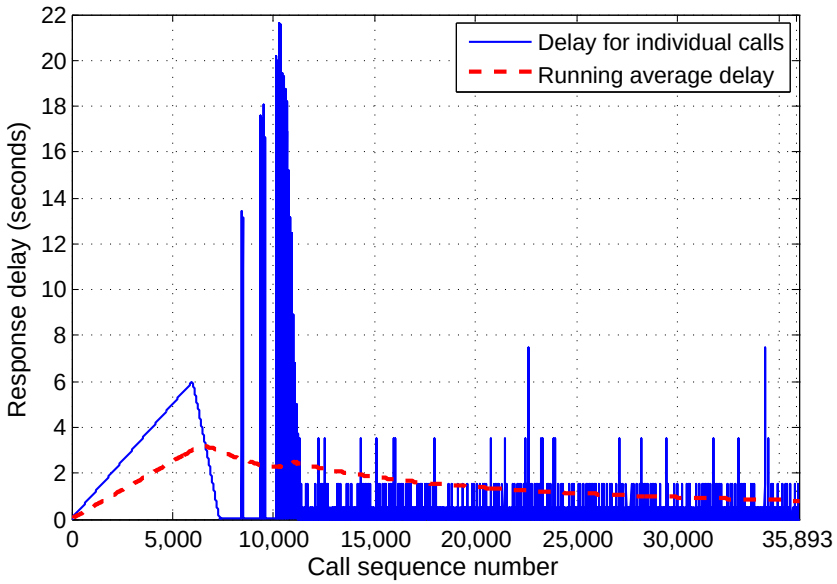


Figure 5.30 : Delay for individual calls in sequence and the running average delay for the PRSM with no removal for loss rate 10%.

6. PERFORMANCE EVALUATION

In this chapter, the performance of an overloaded SIP server equipped with the PRSM has been evaluated through the discrete-time fluid-flow models derived in Chapter 4. Our discrete-event based retransmission analysis and simulation tool developed in Chapter 5 is used to validate the accuracy of the fluid-flow models. To demonstrate the scalability improvement of the PRSM, the conventional SIP design using the corresponding fluid-flow models have also been evaluated. The evaluation is done numerically by implementing the fluid-flow model in Java [65] and using the time slot of 50 ms. For all experiments, the default value of the first retransmission timer T_1 is set to 500 ms as specified in the SIP standard. For all test cases, we assume only INVITE requests responded with 100-Trying.

We conducted three test cases for the performance evaluation. The parameters of first two test cases are used from [8] as a reference for the conventional SIP so that the performance of the PRSM can be compared with the conventional SIP. The third test case is used from Chapter 5 for the performance evaluation under the heavy traffic conditions. These three test cases are as follows:

1. the overload is caused by bulk arrivals while the arrival and service rates are deterministic,

2. the overload is caused by a server slowdown while the arrival and service rates are Poisson distributed,

3. the overload is caused by bursty arrivals which can be modeled with MMPP traffic model.

Sections 6.1.1, 6.2 and 6.3 consist of performance evaluation tests for infinite buffer case, finite buffer case and infinite buffer case with lossy network conditions, respectively.

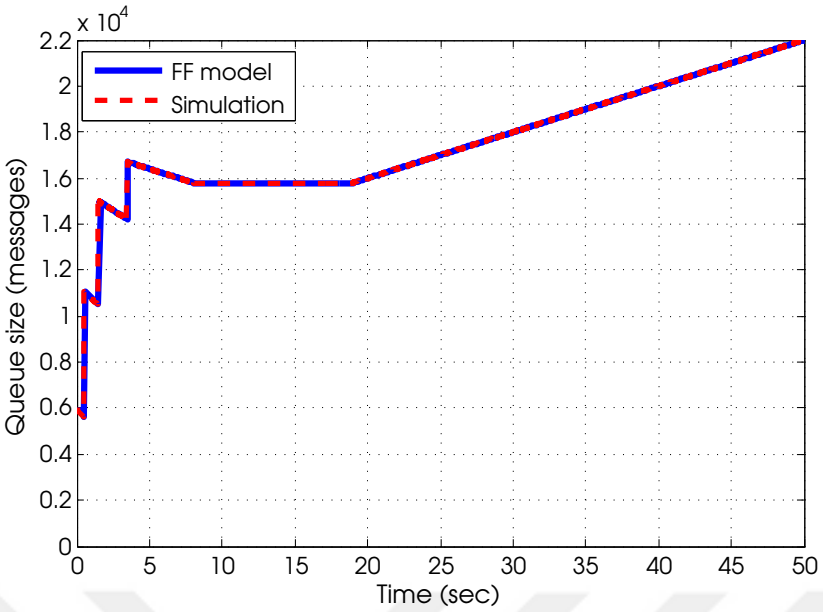


Figure 6.1 : The queue size for the overloaded conventional SIP server when $q(0)$ violates the stability condition.

6.1 Infinite Buffer Case

In this section, we evaluated the performance of the discrete-time fluid-flow model derived in Section 4.6 for infinite buffer case under lossless network conditions.

6.1.1 Overload caused by bulk arrivals

For this test case, bulk messages arrive at time $t = 0s$ overloads the server and are queued (creating an initial queue size), emulating a short surge of user demands. After a demand burst, the original request message arrives at the overloaded server with a constant rate λ=200 messages/sec, emulating a regular user demand. The overloaded server maintains a constant service rate μ=1000 messages/sec. Thus, the effective utilization for the regular user demand is $\rho = \lambda/\mu = 20\%$. Both arrival and service rates are deterministic. The simulation period is 50s.

The reference model in [8] and summarized in Section 4.6.1 defines a stability condition for an initial queue size such that a SIP server can handle the overload effectively under the retransmission mechanism. With the selected arrival and service rates of this test case, the stability condition for the overloaded SIP server is reported as $q(0) < 5700$. Therefore, we consider the case of violating the stability condition, where $q(0) = 6000 > 5700$ messages. Figure 6.1 and Figure 6.2 illustrate the queue

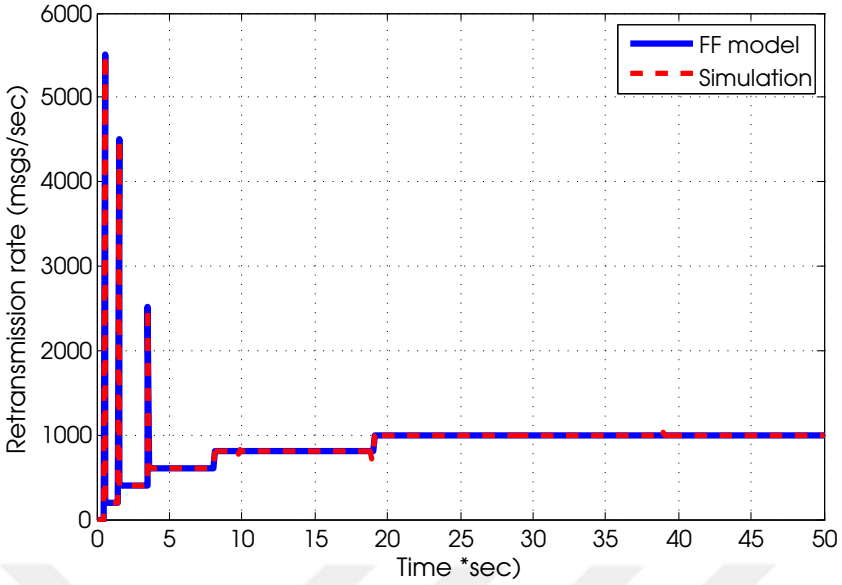


Figure 6.2 : The retransmission rate for the overloaded conventional SIP server when $q(0)$ violates the stability condition.

size and the retransmission rate for the overloaded SIP server, respectively, using both the fluid-flow model and the event-driven simulation tool. First of all, the queue size and the retransmission rate results match with the results of the reference paper [8] indicating the accuracy of our model implementation in Java. Secondly, the results of the fluid-flow model and the event-driven simulation tool match indicating the accuracy of our simulation results.

As detailed in the reference paper [8], after the third-time retransmissions which are triggered at $t = T_3 = 3.5s$, the rate of retransmitted messages increases to 800 messages/sec and the system encounters an aggregated traffic rate of $\lambda + r = 1000$ messages/sec which is equal to the service rate, $\mu = 1000$ messages/sec (as shown in Figure 6.2). At $t = 19s$, the retransmission rate of incoming original messages increases from 800 messages/sec to 1000 messages/sec. Since, after $t = 19s$, the total incoming traffic rate of both original and retransmitted messages is higher than the service rate (i.e., $\rho = 6\lambda/\mu = 1.2 > 1$), the queue size increases linearly with 200 messages/sec (as shown in Figure 6.1). This eventually brings the SIP server crash.

When the same test case is applied to the SIP server with the PRSM, the queue size and the retransmission rate of the server behave differently. Figure 6.3 shows the queue size behavior for the SIP server with the PRSM when $q(0)$ is set to 6000

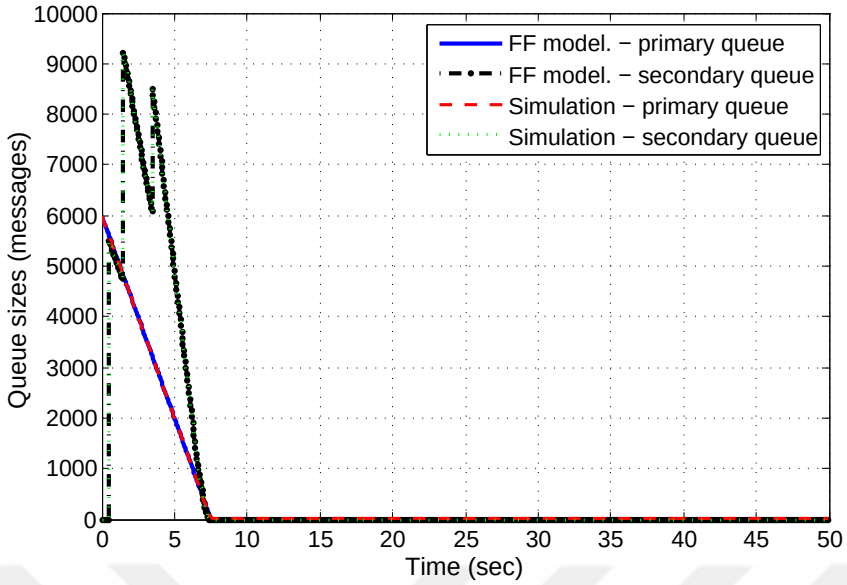


Figure 6.3 : The primary and secondary queue sizes for the overloaded SIP server with the PRSM when $q(0)$ violates the stability condition.

messages (i.e., bulk arrivals). Up to the first retransmission time, $t = 0.5s$, the server has processed 500 messages from bulk arrivals. During this time interval, 100 new requests have joined the primary queue. At $t = 0.5s$, 5500 retransmitted requests join the secondary-queue because of the unprocessed messages of bulk arrivals. After this point, for each processed original message in the primary queue, the retransmission removal process finds a corresponding retransmission message in the secondary queue and removes it.

After $t = 0.5s$, the incoming original messages arrived at $t = 0s$ started to trigger the first-time retransmissions such that 10 retransmissions per time slot. So, up to the second retransmission timer (i.e., $t = 1.5s$), the server processes 1500 messages from the primary queue while 200 retransmission messages join the secondary queue. Therefore, at $t = 1.5s$, with 4500 retransmissions of bulk arrivals, 100 retransmissions of original arrivals (refer to 6.4), the secondary-queue size becomes 8600. Fortunately, after $t = 1.5s$, the server starts to remove two retransmissions (first and second) for each processed original message of bulk arrivals. Similarly, after $t = 3.5s$, three retransmissions (first, second, and third) are removed for each processed original message. The server finishes the processing with bulk arrivals at $t = 6s$, then it starts to process original messages in the primary-queue. At this point, there are

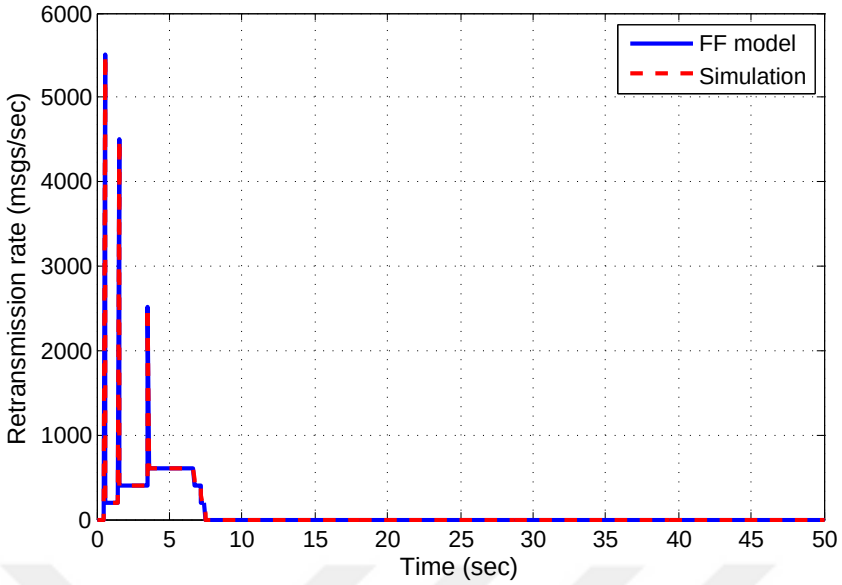


Figure 6.4 : The retransmission rate for the SIP server with the PRSM when $q(0)$ violates the stability condition.

1200 and 2500 messages in the primary and secondary queues, respectively. After $t = 7.35s$, the secondary-queue becomes empty and at this time there are 120 incoming original messages in the primary-queue. Then, the system reaches to the normal processing where the arrival rate $\lambda = 200$ messages/sec and the service rate $\mu = 1000$ messages/sec.

Up to $t = 6s$, the SIP server with the PRSM deals only processing with bulk arrivals and cannot provide any service to incoming original messages. At $t = 0.5s$, the number of aggregated messages consisting of bulk arrivals, incoming original messages, and retransmissions is equal to the queue size of the SIP server. However, after $t = 0.5s$, since the retransmission removal process removes the retransmission messages from the secondary queue, the total number of aggregated messages in the system is lower compared to the SIP server with a single FIFO queue. After $t = 6s$, the SIP server with the PRSM process only incoming original arrivals while the conventional SIP server process both incoming original arrivals and retransmissions. Therefore, the server with the PRSM prevents redundant retransmissions of incoming original arrivals and hence the retransmission rate settles down to 0 around $t = 7.35s$.

To evaluate the effect of the PRSM on the retransmission behavior (and to verify the accuracy of Eqs. (4.13) and (4.15), we also conducted a sub-test case by disabling the

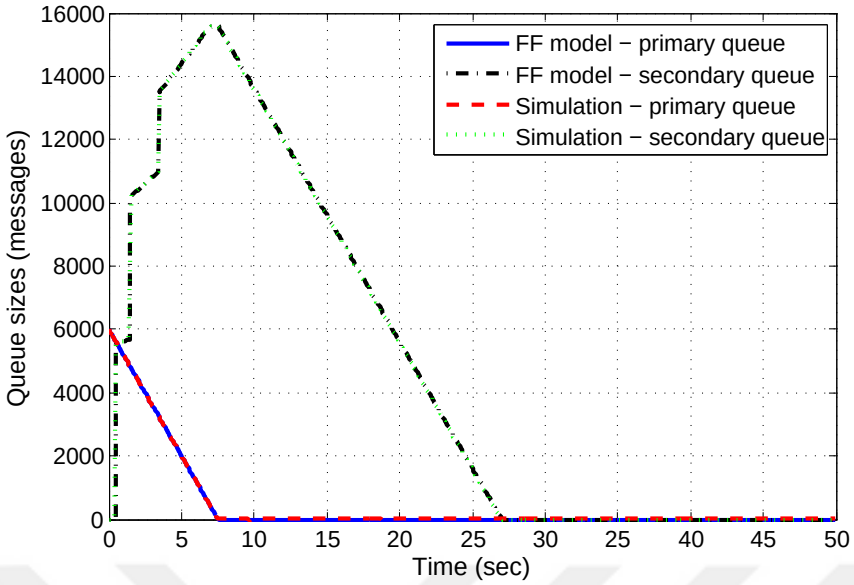


Figure 6.5 : Primary and secondary queue sizes for the SIP server with the PRSM when the removal process is disabled and $q(0)$ violates the stability condition.

retransmission removal process. In this case, retransmission messages in the secondary queue are processed by the remaining processing power of the server according to Eq. (4.13). Figure 6.5 shows the queue size behavior for the SIP server with the PRSM while the removal process is disabled. Since the server encounters bulk arrivals at the beginning, it cannot process retransmissions in the secondary queue until processing of bulk messages and new incoming original arrivals. The server starts having idle time for processing the secondary-queue with the rate of 800 message/sec at $t = 7.5s$. Since there are 15610 retransmitted messages in the secondary-queue up to $t = 7.5s$, the processing time takes $15610/800 = 19.5s$, and hence the secondary-queue becomes empty around $t = 27s$. Note that enabling or disabling the removal process does not affect the primary queue size, and therefore the retransmission rate behavior is observed to be exactly the same in both cases (Figure 6.4).

As evaluated in Section 5.5, by providing higher priority to original requests, the SIP server with the PRSM limits the first retransmissions and hence the number of subsequent retransmissions. When the removal process is enabled, the server removes redundant retransmissions timely and hence becomes prepared for subsequent bulk arrivals.

6.1.1.1 Test duration for 150 seconds

For this test case, the number of failed call attempts is not recorded for both the conventional SIP and the PRSM, since the current traffic settings creates only three retransmissions for each call attempt. while the queue size is continuously increasing, we expected to record some call failures for the conventional SIP server case. For this reason, we extended the duration of the test case to 150 seconds for the conventional SIP server case. As shown in Figure 6.6, in this extension, we also reported the response delay.

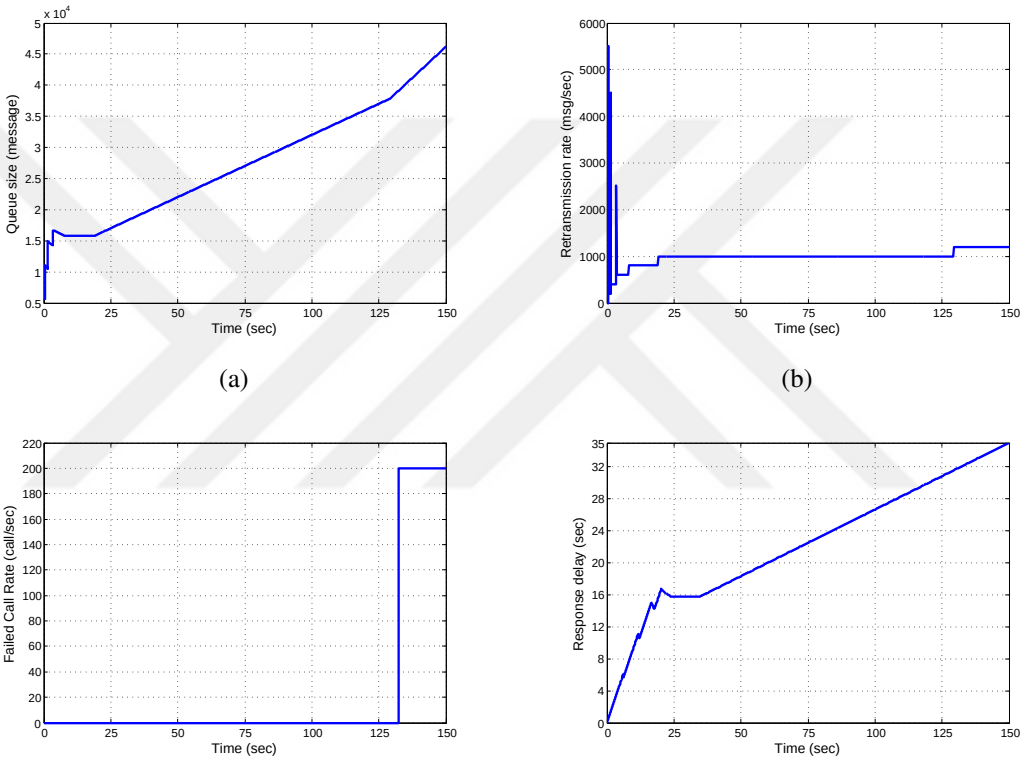


Figure 6.6 : Results from the extended test case-1 as changes in queuing dynamics (a) queue size, (b) retransmission rate, and performance metrics (c) failed call attempts, (b) response delay, versus time.

Figure 6.6 shows the results from the extended version of the test case-1, for various queuing dynamics and performance metrics, the number of failed call attempts and response delay versus the time. Since, the aggregated arrival rate (original arrival rate + retransmission rate) reaches 1200 messages/sec. (200 + 1000) at t=19 seconds, the queue size increases linearly with 200 message/sec (refer to (a) for queue size and (b) for retransmission rate). As the effect of increasing in the queue size, the response delay also starts to increase almost linearly with 200 ms/sec., from t=35 seconds (refer

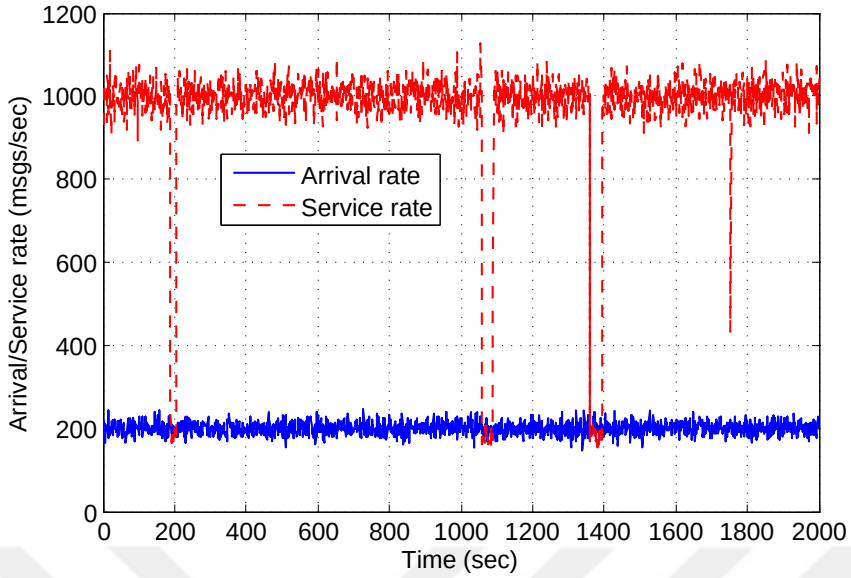
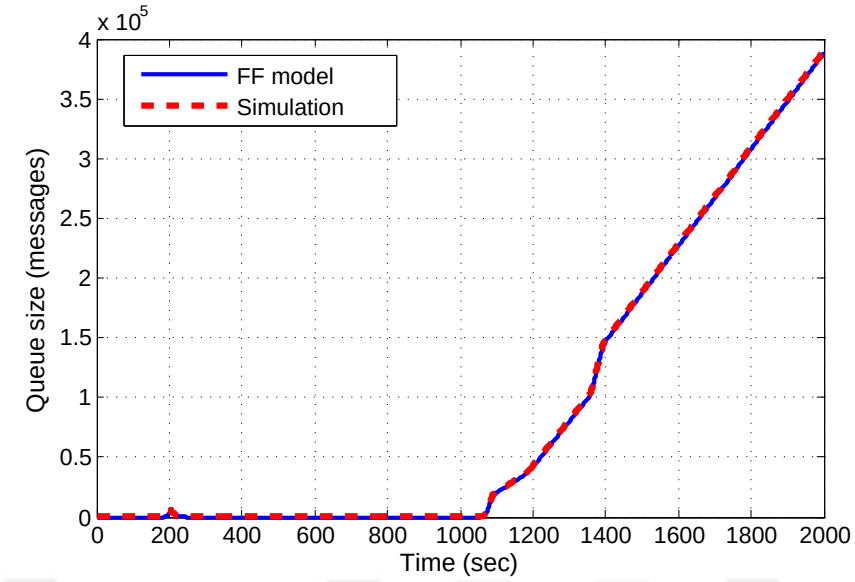


Figure 6.7 : Service and original message arrival rates for overloading SIP server performing normal and maintenance mode services alternately.

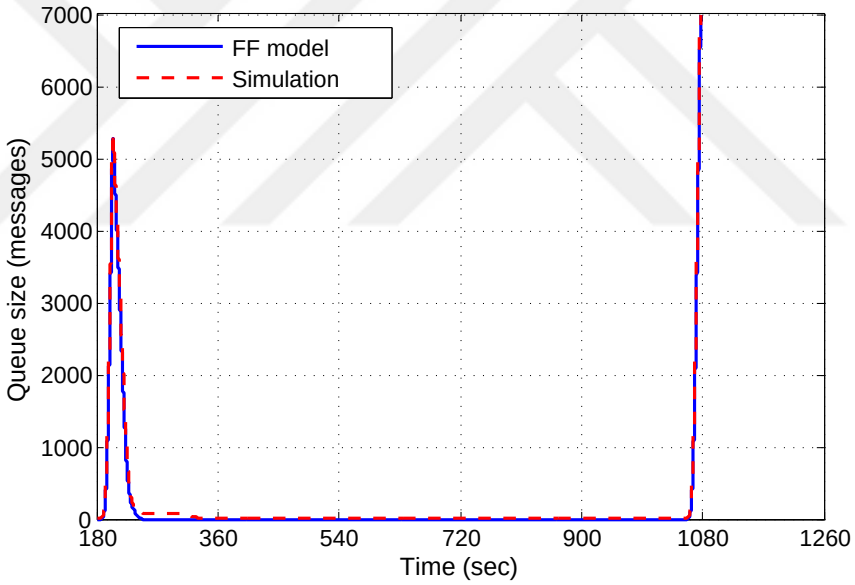
to (d)). At t=129.35 seconds the retransmission rate reaches to 1200 message/sec. so that the increase rate of the queue size reaches to 400 message/sec., and at t=132.05 seconds the response delay starts to exceed 32sec. (64x500ms=32sec). As the result, call failures start to occur at t=132.35 seconds with a constant rate of 200 message/sec. (refer to (c)).

6.1.2 Overload caused by temporal server slow down

In this test case, the SIP server works in one of two alternating modes: normal service or maintenance. Because of the reduced server processing capacity during the maintenance period, the server may be overloaded. The mean arrival rate is $\lambda = 200$ messages/sec; the mean service rate at the normal service mode $\mu_1 = 1000$ messages/sec; the mean service rate at the maintenance mode is $\mu_2 = 180$ messages/sec; all are Poisson distributed. In order to have a similar distribution of mode transitions used at [8], the durations of the normal service and maintenance modes are set to 188.0s and 15.7sec, 854.9s and 30.9sec, 270.2s and 34.4sec, and 358.2s and 1.4sec, respectively. With this transition distribution, the first maintenance mode duration is short enough such that the conventional SIP server can cope with the original and retransmitted messages without resulting in a server crash. The following



(a) Normal view



(b) Enlarged partly view

Figure 6.8 : Queue size versus time for the overloaded conventional SIP server which performed normal service and maintenance service alternately.

two maintenance mode durations creates high queue sizes (as shown in Figure 6.7). The simulation period is 2000 seconds and hence the last duration of the normal service mode runs up to 2000 seconds. We repeated the test case 100 times for both fluid-flow model and simulation methods and the numerical results are reported as the average values of 100 experiments for each method.

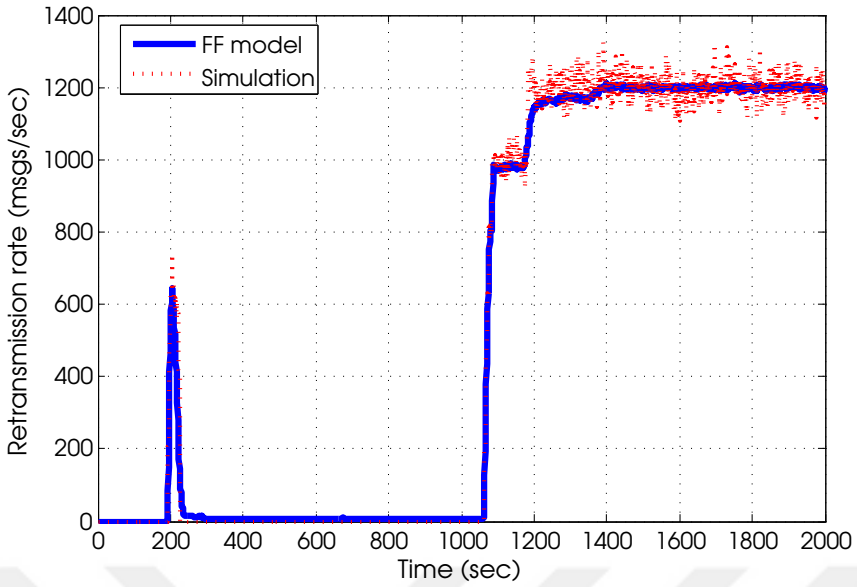
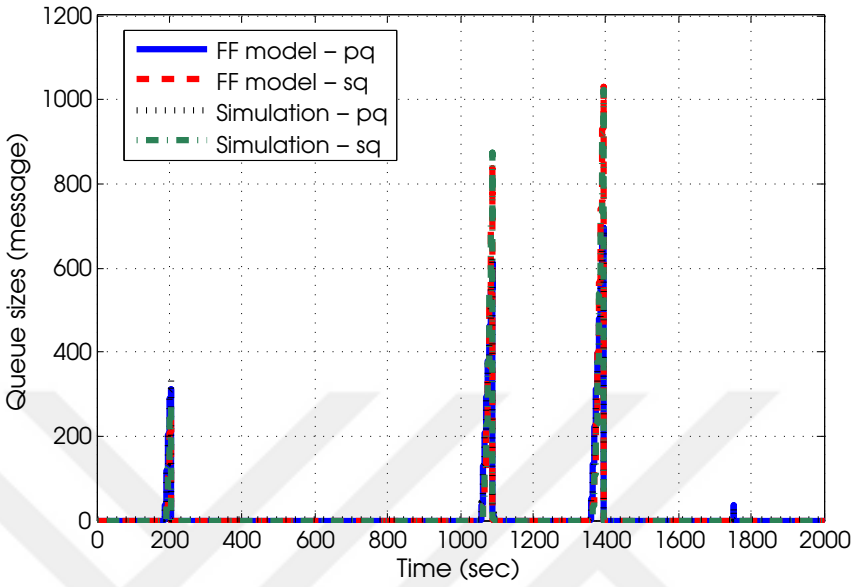


Figure 6.9 : The retransmission rate for the overloaded SIP server which performs normal and maintenance service modes alternately.

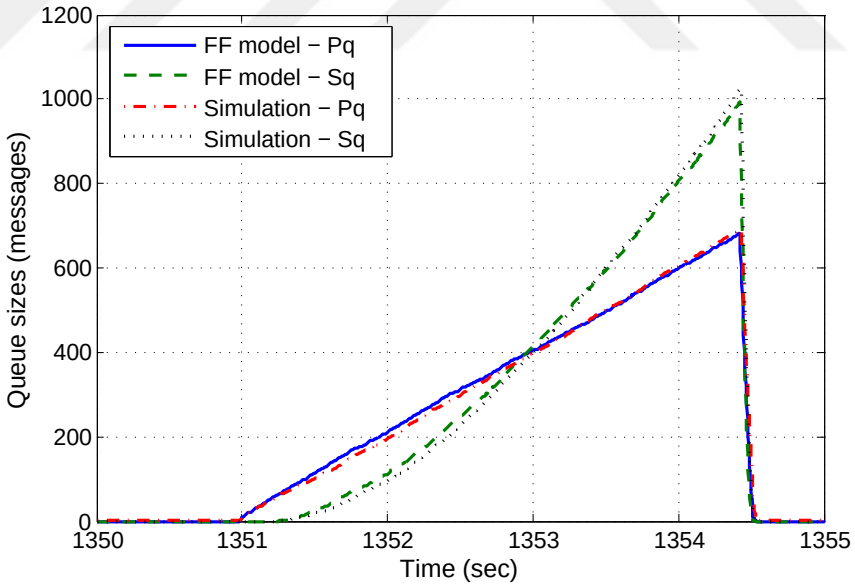
As shown in Figure 6.7 and Figure 6.8, the first maintenance mode duration is short enough such that the conventional SIP server can cope with the original and retransmitted messages without resulting in a server crash. The following two maintenance mode durations creates high queue sizes.

Figure 6.8 and Figure 6.9 illustrate the queue size and retransmission rate for the conventional overloaded SIP server, respectively. As shown in Figure 6.8, the server enters the maintenance mode almost at $t = 188s$ for $15.7s$. The messages start to accumulate in the queue and reach a peek around 5100 messages in average which obeys the stability condition 5700 messages. After this maintenance mode, the server can process all accumulated messages in time and the queue size decreases. The second maintenance mode occurs at $t = 1058.6s$ for around $30.9s$. This relatively long maintenance period creates high queue sizes. The subsequent maintenance modes make the situation worse and the queue size tends to steadily increase by the retransmissions. The stability condition is violated and hence the SIP server will be eventually crashed.

Figure 6.10 illustrates the queue size when the PRSM is enabled. As shown in Figure 6.10, the server with the PRSM copes with the overload situation by keeping the queue sizes of both the primary and secondary queues below the stability condition (e.g., the



(a) Normal view



(b) Enlarged partly view

Figure 6.10 : Primary and secondary queue sizes for the overloaded SIP server with the PRSM, which performs normal and maintenance service modes alternately.

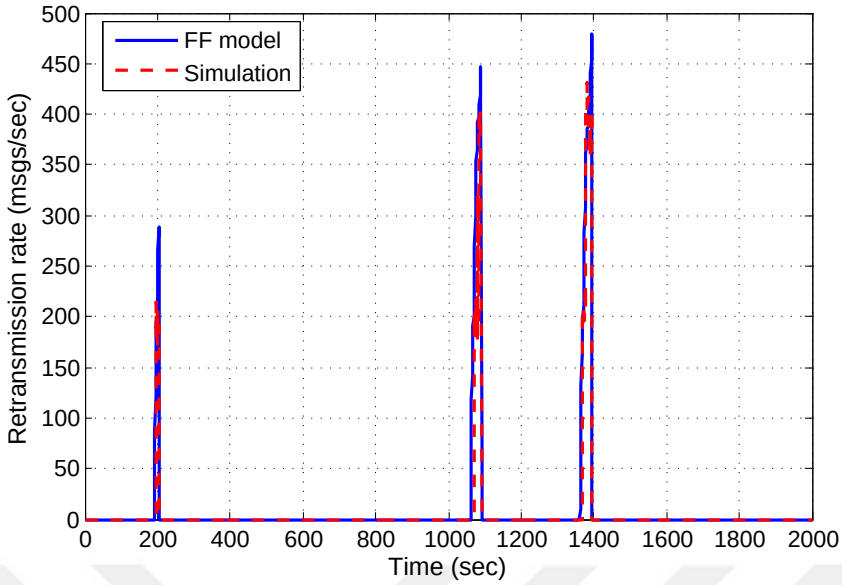


Figure 6.11 : The retransmission rate for the overloaded SIP server with the PRSM which performs normal and maintenance service modes alternately.

secondary queue size is at most 1023 while the primary queue size is at most 690). For the most problematic maintenance mode (started around $t = 1359.7s$, exited around $t = 1394.1s$), the total number of messages in the primary and secondary queues is kept the peak value of around 1700 messages while the number of messages in the conventional SIP server around 145900. The retransmission rate behaved similar to the queue size behavior (see Figure 6.11) over the observed time scale and reached a peak value of 450 messages/sec while it is around 1200 messages/sec for the overloaded SIP server. For the conventional SIP, the average value of the ratio of the total number of failed attempts to the total number of arrivals is 0.396 for the fluid flow model and 0.399 for the simulation. No call failure is observed for the PRSM.

6.1.3 Overload caused by bursty arrivals

This test case employs Markov Modulated Poisson Process (MMPP) which is heavily used to model bursty traffic for telecommunication networks. In a typical 2-mode MMPP, denoted MMPP(2), traffic model, a single exponential service rate and two different arrival rates (overloading and non-overloading) are considered to create the bursty traffic arrival pattern. In the overloading mode, the model uses a Poisson process with an INVITE request arrival rate λ_0 for a period of time that is exponentially distributed with the parameter δ_0. Then, it moves to the non-overloading mode, in

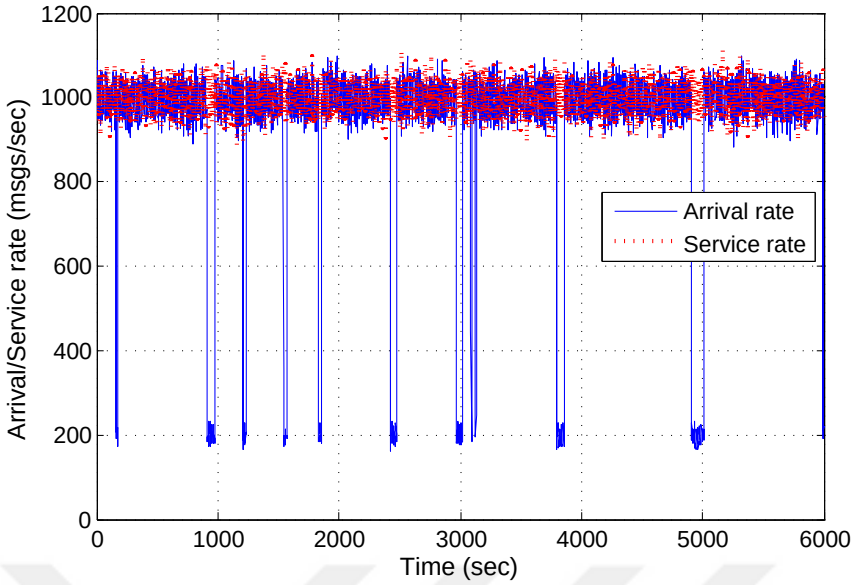


Figure 6.12 : Original message arrival rate and server service rate for pseudo-random mode transitions of MMPP.

which the traffic model uses a Poisson process with the arrival rate λ_1 for a period of time that is exponentially distributed with the parameter δ_1, where δ_0 and δ_1 represent mode duration parameters. Note that smaller δ values yield higher mean queue size and blocking probability [57].

This test case uses an MMPP(2) traffic model with the mode parameters $\delta_0 = 0.00195$ and $\delta_1 = 0.01953$ to simulate heavy traffic conditions. The mean arrival rate for the heavy traffic mode is set to $\lambda_0 = 995$ messages/sec; the mean arrival rate for the medium traffic mode is set to $\lambda_1 = 199$ messages/sec; the mean service rate is chosen as $\mu = 1000$ messages/sec. In order to have a comparable base, we used pseudo-random distributions for mode transitions by using the seed 0 for random number generators for both distributions. (Note that Java programming language has a built-in random number generator, which uses a 48-bit seed using a linear congruential formula, provides identical random sequences with the same seed as referring to Section 3.2.1 of [66]. Refer to Java language `Random` class definition for details). This approach produced an arrival rate distribution as shown in Figure 6.12. The simulation time is set to 6000 seconds. Each test case is repeated 100 times for both the fluid-flow model and the simulation tool and the results are reported as the average values of 100 experiments.

For this test case, as we already observed in Section 5.5, the simulation tool crashed for the conventional SIP server, since there are no enough computing resources for very long queue sizes due to the excessive number of retransmissions. Therefore, we cannot provide the numerical results for the conventional SIP server using the simulation tool. The numerical results from flow-fluid model shows that heavy traffic conditions create high retransmission rate (Figure 6.13) so lead to linearly increasing queue size (refer to Figure 6.14). This indicates that this increase in the queue size may eventually cause the server crash. For the conventional SIP, the average value of the ratio of the total number of failed attempts to the total number of arrivals is 0.9195 as obtained from the fluid flow model. Figure 6.15 shows the primary and secondary queue sizes for

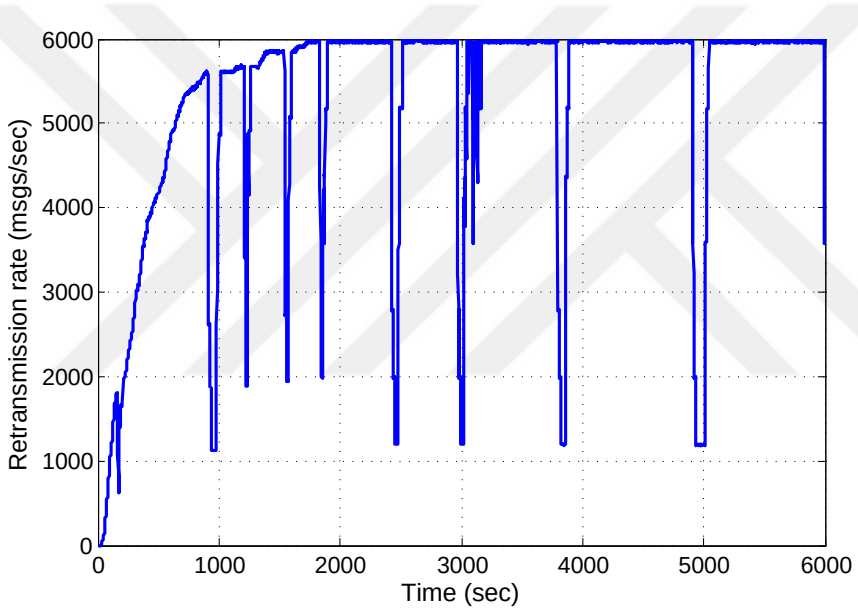


Figure 6.13 : Change of the retransmission rate for the conventional SIP server overloaded by bursty arrivals (fluid-flow model).

the SIP server with the PRSM which is overloaded by bursty arrivals. The server can keep the primary queue almost at 200 messages and the secondary queue size at a low level during burty periods by keeping the retransmission rate at 50 messages/second in average. The similarity between the numerical results of the fluid-flow model and the simulation tool demonstrates that the fluid-flow model can accurately capture the behavior of the PRSM. We do not observed failed attempts in this test case, for the PRSM.

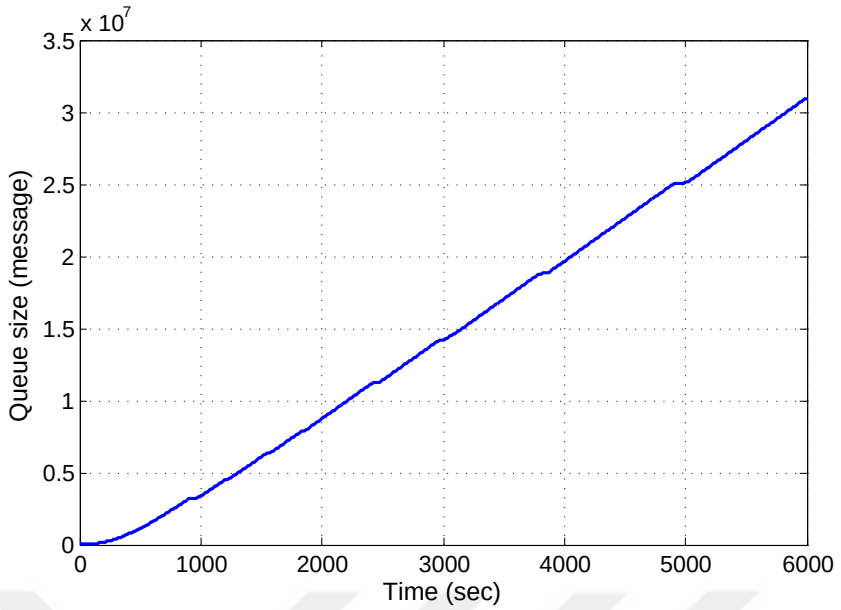


Figure 6.14 : Change of the queue size for the conventional SIP server overloaded by bursty arrivals (fluid-flow model).

6.1.4 PRSM stability condition test

This test case is conducted to verify the stability conditions of the PRSM defined at Section 4.6.2.3. For this purpose, we conducted the test case with condition that $\lambda > \mu$. The test case employs a 2-mode MMPP traffic model with the mode parameters as $\delta_0 = 0.00195$ and $\delta_1 = 0.01953$ to simulate heavy traffic conditions. The mean arrival rate for the heavy traffic mode λ_0 is set to 995, 1045, 1095, 1145, and 1195 messages/sec to obtain conditions providing both $\lambda < \mu$ and $\lambda > \mu$ situations. The mean arrival rate for the medium traffic mode is set to $\lambda_1 = 199$ messages/sec; the mean service rate is chosen as $\mu = 1000$ messages/sec. We used the setting of infinite buffer for the server to eliminate the blocking effect of the finite buffer. We used Java implementation of the fluid-flow model of the PRSM to obtain results, such that, as similar to the test case at Section 6.1.3 to have a comparable base, the pseudo-random distributions for mode transitions by using the seed 0 for random number generators for both distributions. The time for the test case is set to 6000 seconds. We repeated test case 100 times for both the fluid-flow model and the simulation tool and results are reported as average values of 100 experiments.

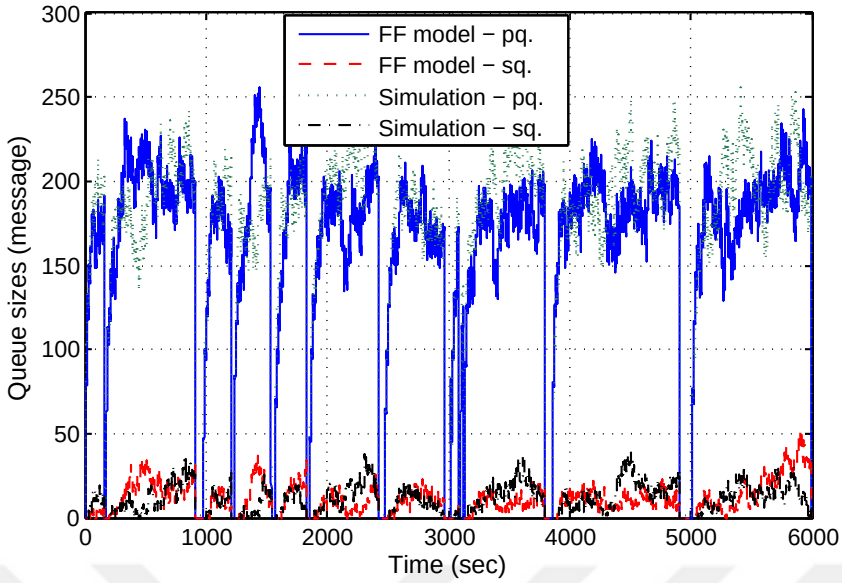
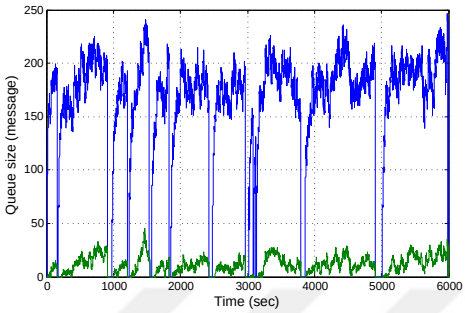


Figure 6.15 : Primary and secondary queue sizes for the server with the PRSM which is overloaded by bursty arrivals.

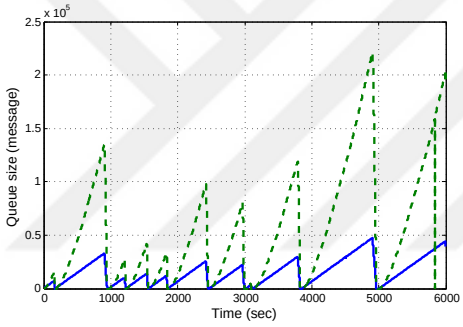
Table 6.1 : Call failure ratio comparison with various λ_0 values.

λ_0	**995**	**1045**	**1095**	**1145**	**1195**
Call failure ratio	0	0.107	0.813	0.939	0.954

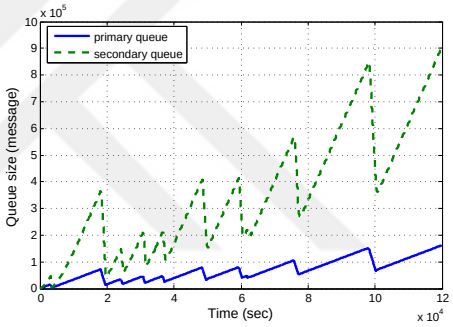
The results are given in the Figures 6.16, 6.17 and 6.18 and the Table 6.1. As shown in the figures, as the arrival rate of the heavy traffic mode λ_0 increased, queue size in the server starts to increases almost linearly except for the first 5.02% increase, $\lambda_0 = 1045$ (refer to Figure 6.16), since the retransmission rate increases through 8K message/sec (refer to Figure 6.17). The response delay is also increased from 4 seconds in max for $\lambda_0 = 995$ to 600 seconds in max for $\lambda_0 = 1195$ (refer to Figure 6.18). The mean value of the call failure ratio reached to 0.954 for $\lambda_0 = 1195$, while is zero for $\lambda_0 = 995$ and 0.107 for $\lambda_0 = 1045$ (refer to Table 6.1). Theoretically, since the PRSM eliminates the effect of redundant retransmissions and under the assumptions that there is no network loss and the server has infinite buffer size, the server equipped with the PRSM behaves like an M/M/1 queue encountered the traffic of original arrivals. Therefore, the stability condition of the PRSM is the same as the stability condition definition of a general queue with a single server and infinite buffer: $\rho = \lambda/\mu < 1$. For MMPP, we can calculate the average arrival rate that the server will encounter, given by Eq.(5.1). For the parameters of this case, the server can cope with an $\lambda_{av} < 1000$, and therefore $\lambda_0 < 1079,98$. Table 6.2 gives λ_{av} values for this test case.



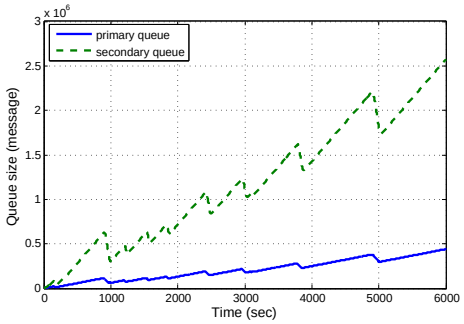
(a) $\lambda_0 = 995$



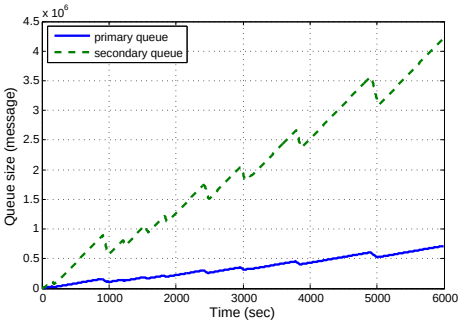
(c) $\lambda_0 = 1045$



(d) $\lambda_0 = 1095$

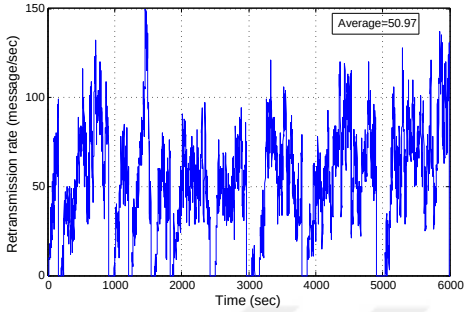


(e) $\lambda_0 = 1145$

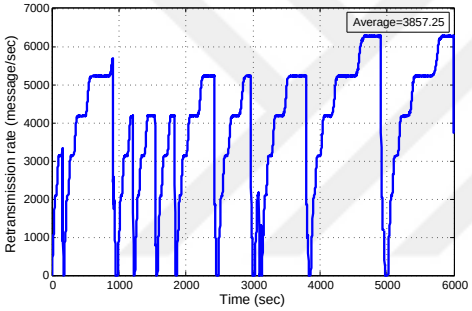


(f) $\lambda_0 = 1195$

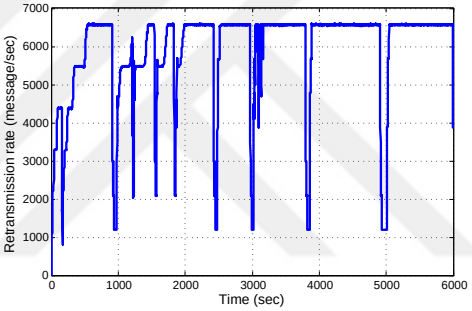
Figure 6.16 : Queue size versus time for λ_0 settings 995, 1045, 1095, 1145 and 1195.



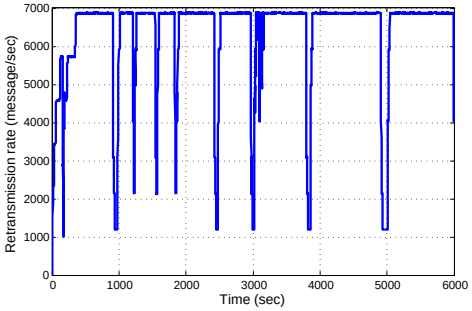
(a) $\lambda_0 = 995$



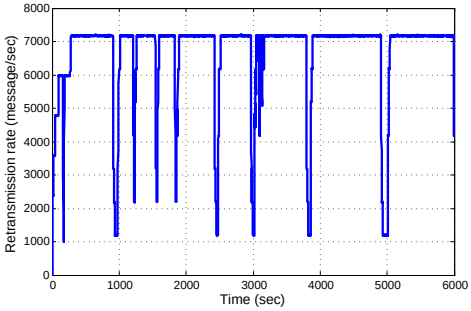
(c) $\lambda_0 = 1045$



(d) $\lambda_0 = 1095$

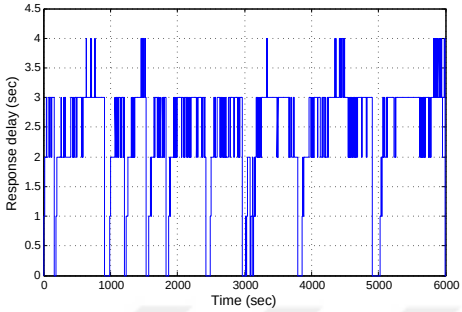


(e) $\lambda_0 = 1145$

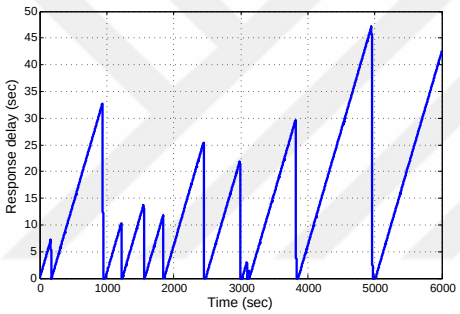


(f) $\lambda_0 = 1195$

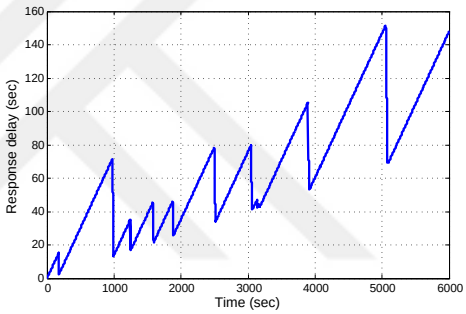
Figure 6.17 : Retransmission rate versus time for λ_0 settings 995, 1045, 1095, 1145 and 1195.



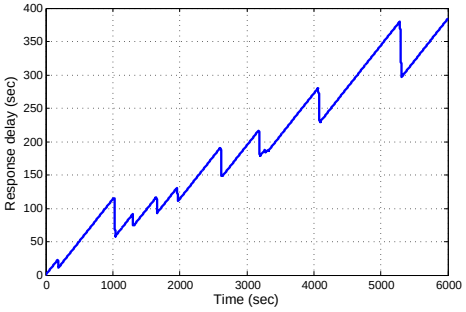
(a) $\lambda_0 = 995$



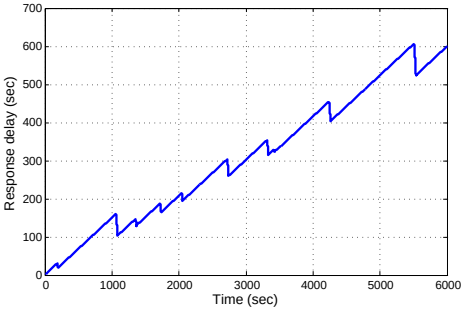
(c) $\lambda_0 = 1045$



(d) $\lambda_0 = 1095$



(e) $\lambda_0 = 1145$



(f) $\lambda_0 = 1195$

Figure 6.18 : Response delay versus time for λ_0 settings 995, 1045, 1095, 1145 and 1195.

Table 6.2 : Average arrival rates for selected λ_0 values.

λ_0	**995**	**1045**	**1095**	**1145**	**1195**
λ_{av}	922.74	968,20	1013,65	1059,12	1104,58

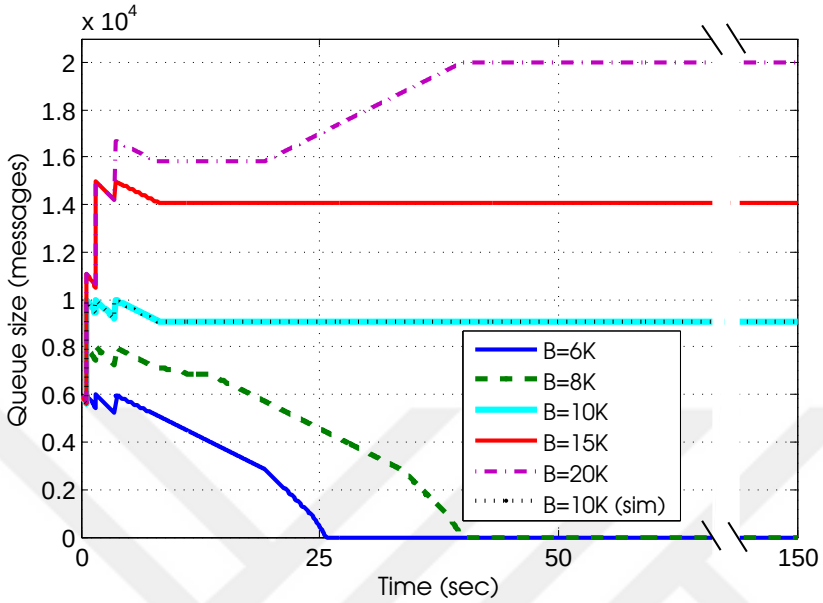


Figure 6.19 : Queue size change for the conventional SIP server when the initial queue size is set to 6K messages, for various buffer sizes, B.

6.2 Finite Buffer Case

In this section, we evaluated the performance of the discrete-time fluid-flow model derived in Section 4.7 for finite buffer case under lossless network conditions.

6.2.1 Overload caused by bulk arrivals

Recalling settings from Section 6.1.1, for this case the extension is that the simulation period is 150 seconds.

For the infinite buffer case, [8] defines a stability condition for an initial queue size such that a SIP server can handle the overload effectively under the retransmission mechanism. With the selected arrival and service rates of this test case, the stability condition for the overloaded SIP server is reported as $q(0) < 5.7K$. Therefore, we consider the case of violating the stability condition, where $q(0) = 6K > 5.7K$ messages. For the finite buffer case, we used buffer size assignments as $6K$, $8K$, $10K$,

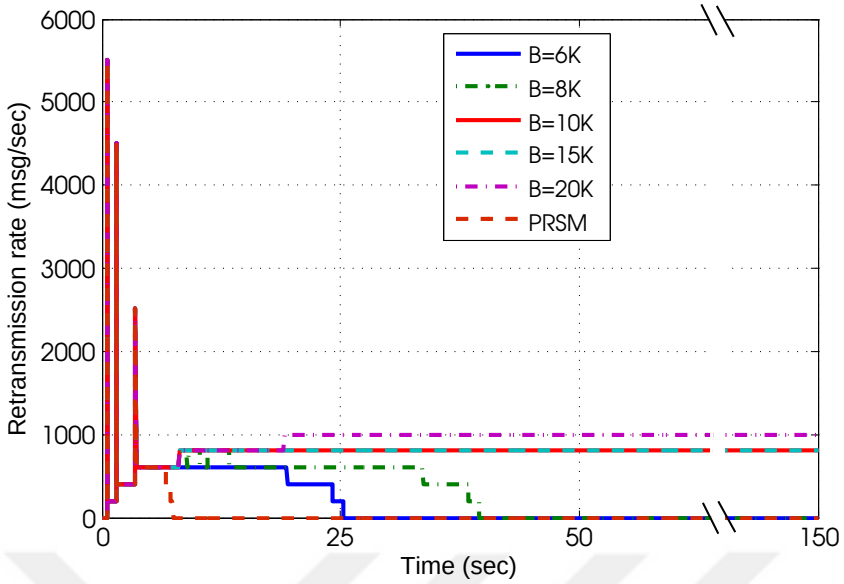


Figure 6.20 : Change of the retransmission rate for the conventional SIP server and for the PRSM when the initial queue size is set to 6K messages, for various buffer sizes, B.

$15K$ and $20K$. As demonstrated for the buffer size of 20K in Figure 6.19, $6K$ initial messages create three bulk retransmissions as $5.5K$ at $t = 0.5s$, $4.5K$ at $t = 1.5s$ and $2.5K$ at $t = 3.5s$, such that with additional new arrivals and retransmissions, the queue size reaches a peak around $16.7K$ at $t = 3.5s$. The conventional SIP server can cope the overload condition for smaller buffer sizes ($B = 6K$ and $B = 8K$) by rejecting arrivals exceeding the buffer size, and therefore both queue sizes and retransmission rate reach to zero as shown in Figure 6.19 and Figure 6.20, respectively. For $B = 20K > 16.7K$, the server accepts all retransmissions into the queue and is obliged to process them. Therefore, the retransmission rate reaches to 1000 msgs/sec at $t = 19s$ (refer to Figure 6.20), which is the processing capacity of the server. Then, the queue size steadily grows and eventually reaches to the full buffer situation. For $B = 10K$ and $B = 15K$, the retransmission rates are limited to 800 msgs/sec and therefore the queue sizes remains constant around the full buffer sizes. Note that, there is no failed call attempt recorded for all buffer size cases. Figure 6.19 also illustrates the queue size behavior from the simulation tool for $B = 10K$ and other simulation results are omitted for the sake of presentation. The results of the simulation tool and the fluid-flow model match closely indicating the correctness of the fluid-flow model.

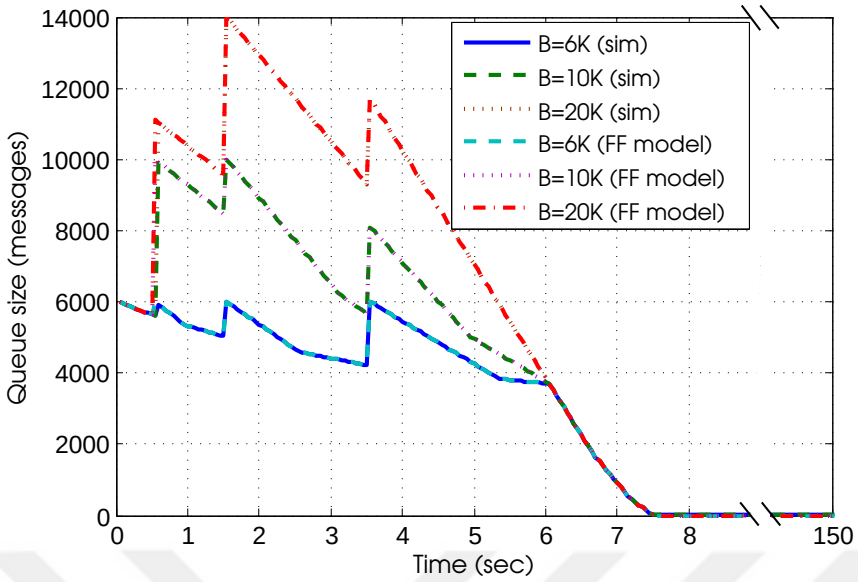


Figure 6.21 : PRSM queue size change for buffer sizes, 6K, 10K and 20K for fluid-flow model and simulation.

Figure 6.19 also includes the queue size results of the simulation tool for $B = 10K$. The results of the simulation tool and the fluid-flow model match closely indicating the accuracy of fluid-flow model.

Figure 6.21 shows the queue size behavior of the SIP server with the PRSM for both fluid-flow model and simulation with the same traffic conditions and $6K$, $10K$ and $20K$ buffer sizes. Without depending on the buffer size, $t = 0.5s$, the server has processed 500 messages from bulk arrivals up to the first retransmission time. During this time interval, 100 new requests have joined the primary queue. Then, while the total queue size is $5.6K$, $5.5K$ retransmitted requests arrive at the server because of the unprocessed messages of initial bulk arrivals. While the server equipped with buffer sizes of $6K$ and $10K$ reject a part of retransmissions, the server with $20K$ accepts all retransmitted messages and the queue size reaches to $11.1K$. After this point, for each processed original message in the primary queue, the retransmission removal process finds related retransmission messages in the secondary queue and removes them. Besides, after $t = 0.5s$, original new arrivals also start 1st-time retransmissions with retransmission arrival rate 200 messages/sec (refer to Figure 6.20 for the PRSM). At $t = 1.5s$, the server encountered with $4.5K$ bulk retransmissions from initial arrivals while the total queue size $9.5K$ ($11.1K - 1000 - 1000 + 200 + 200$), then the total

queue size becomes $14K$ for the server with 20K buffer. After that time, removal process starts to remove two retransmissions for each processed original message from the initial arrivals. Besides, 2^{nd}-time retransmissions start for new arrivals, then the retransmission arrival rate becomes 400 messages/sec. Similarly, at $t = 3.5s$, 3^{rd}-time retransmissions arrive at the server $2.5K$ for initial arrivals, and then the retransmission arrivals rate reaches 600 messages/sec. After $t = 6s$, the server finishes the processing initial arrivals while the total queue size is $3.7K$, and starts to process new arrivals. This decreases the retransmission arrival rate step-by-step (refer to Figure 6.20 for the PRSM). At $t = 7.5s$, both the primary and secondary queues become empty and the server reaches the normal processing situation. The retransmission rate does not exceeds 600 messages/sec and is fading to zero almost at $t = 7.5s$, by giving higher priority to the original requests.

As a summary, the server can cope with the overload conditions as prioritizing the original requests and retransmissions and using the removal process even if the buffer is higher enough to accept all retransmissions into the queue. The results included from simulation tool illustrate the accuracy of the fluid-flow model provided for the PRSM. Note that we observed the same behavior for all buffer size cases covered in this test case, and we only report the results of three buffer sizes for the sake of presentation.

6.2.2 Overload caused by temporal server slow down

We recall test case parameter settings from Section 6.1.2, such that arrival and service rates shown in Figure 6.7 are valid for this test.

As shown in Figure 6.22, the first maintenance mode duration is short enough such that the conventional SIP server can cope with the original and retransmitted messages without resulting in high queue sizes. The following two maintenance mode durations creates high queue sizes for higher buffer sizes. For $B = 20K$, the queue continuously stays in full buffer situation. The figure consists of the queue size results of the simulation tool for indicating the accuracy of fluid-flow model results.

Figure 6.23 shows the retransmission rate behavior of the conventional SIP server by using the results from the fluid-flow model. The retransmission arrival rate reaches almost 1200 messages/sec at the third maintenance period for higher buffer sizes.

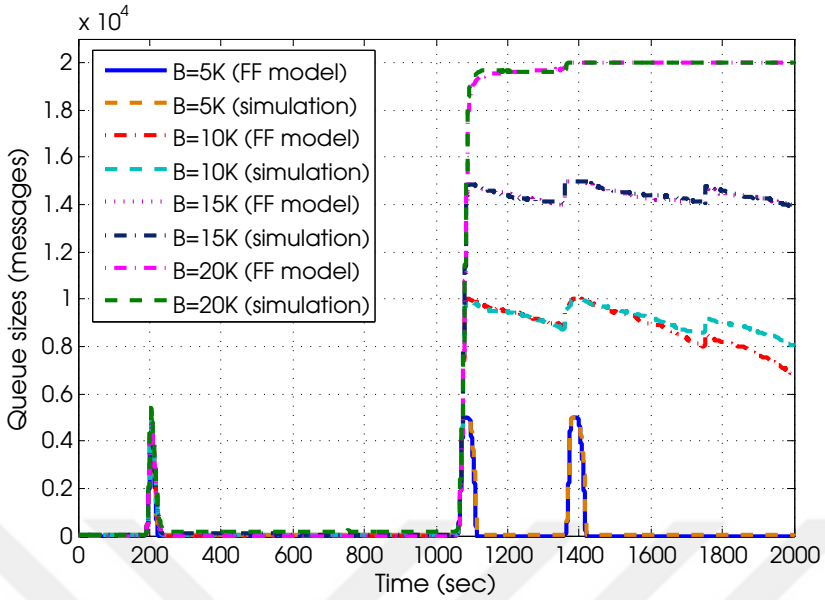


Figure 6.22 : Queue size change for the overloaded SIP server which performs normal and maintenance service modes alternately (model and simulation).

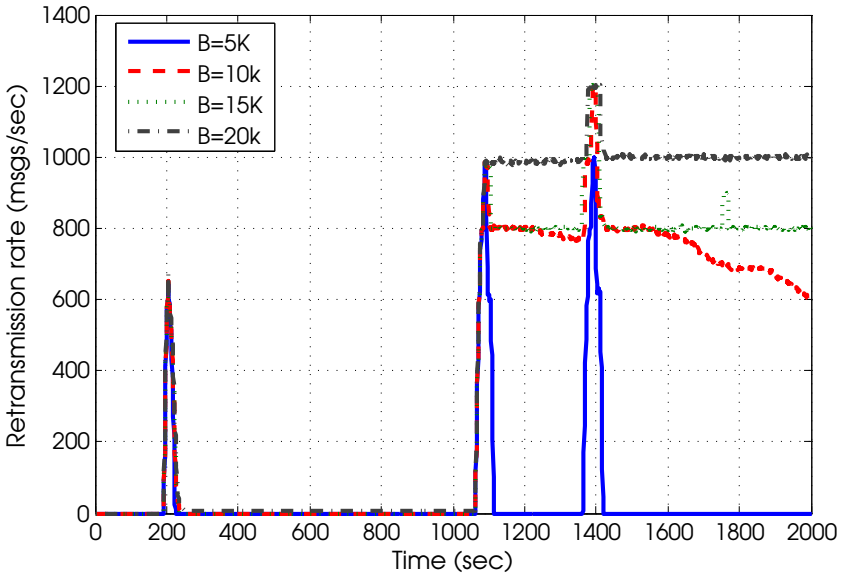


Figure 6.23 : Change of the retransmission rate for the conventional SIP server which performs normal and maintenance service modes alternately (fluid-flow model).

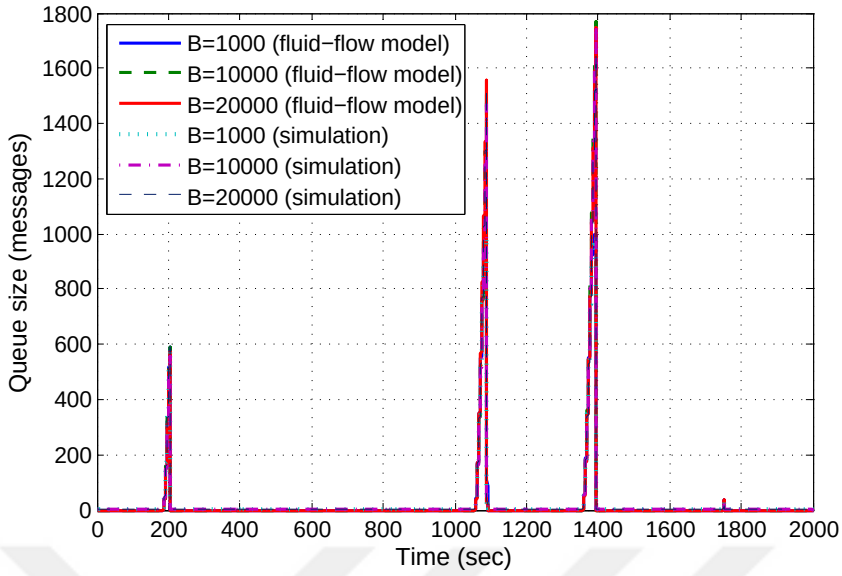


Figure 6.24 : Queue size change for SIP server with PRSM which performs normal and maintenance service modes alternately.

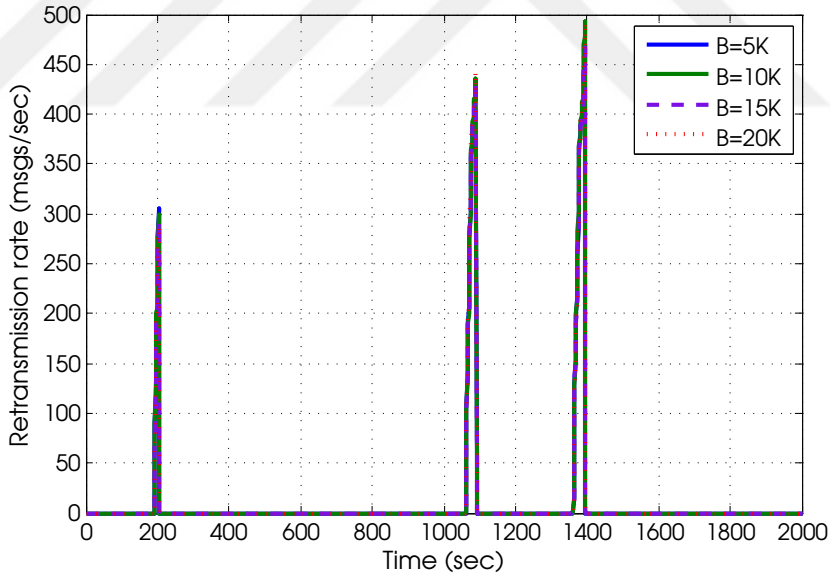


Figure 6.25 : Retransmission rate change for SIP server with PRSM which performs normal and maintenance service modes alternately.

Figure 6.24 illustrates the queue size when the PRSM is enabled. The server with the PRSM copes with the overload situation by keeping the total queue size as around $1.7K$ for higher buffer sizes for the third maintenance period. As shown in Figure 6.25, the retransmission rate behavior is similar to the queue size behavior over the observed

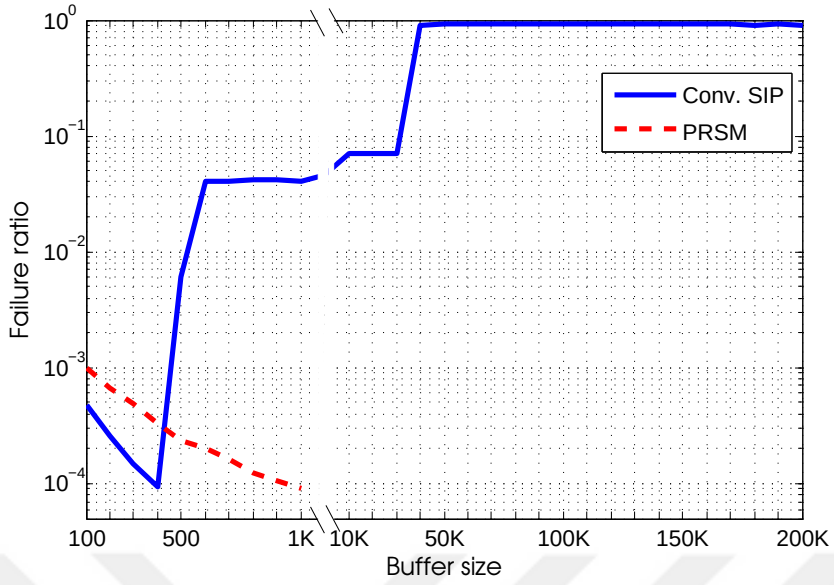


Figure 6.26 : The mean ratio of the number of failed call attempts to the number of all call attempts for various buffer sizes.

time scale and reached a peak value of 490 messages/sec while it is around 1200 messages/sec for the conventional SIP server. For the conventional SIP, the average value of the ratio of the total number of failed attempts to the total number of arrivals is 0.396 for the fluid flow model and 0.399 for the simulation. There is no call failure observed for the PRSM.

6.2.3 Overload caused by bursty arrivals

By recalling test case settings from Section 6.1.3, in this test case, we reported the failed call attempt and 100-Trying response delay behaviors for both the conventional SIP and PRSM. Since the accuracy of the fluid-flow model is validated in the previous test cases, in this test case, we will not include the results from the simulation.

Figure 6.26 and Figure 6.27 show the mean ratio of the number of failed call attempts to the number of all call attempts and the mean delay for sending 100-Trying response for various buffer sizes, respectively for both the conventional SIP and PRSM (note that the y-axis is logarithmically scaled for both figures). Under the assumption of lossless network conditions, there are two main factors for retransmissions: the blocking probability and the queuing delay. For smaller buffer sizes, the blocking probability is effective on retransmissions and these retransmissions are non-redundant. When

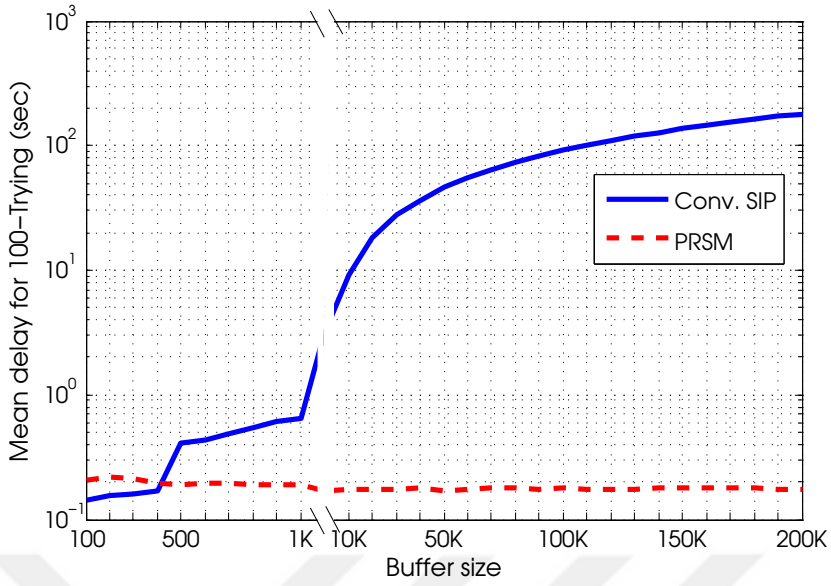


Figure 6.27 : The mean delay for sending 100-Trying response for various buffer sizes.

the queuing delay starts to exceeds T_1 (i.e. $q > \mu T_1$) retransmissions are triggered. These retransmissions are redundant and negatively affect the processing capacity of the server for the conventional SIP. With the parameters of $T_1 = 0.5sec$ and $\mu = 1000$ messages/sec, for $B \geq 500$, the queuing delay becomes effective and the mean failure ratio and the mean response delay take higher values compared to the smaller buffer size cases. In other words, when B is smaller than 500, the blocking probability becomes a dominating factor on retransmissions. Higher B values may bring additional retransmissions because of higher queuing delay.

For PRSM, although, for $B < 500$, the failure ratio and response delay have higher values; for higher buffer sizes, both metrics take smaller values. The failure ratio for the server with PRSM reaches to zero for $B > 1K$, while the conventional SIP reaches a value around 0.9 for $B > 40K$. Similarly, the response delay for PRSM is around 180ms for $B > 1K$, while the conventional SIP continuously increases when the buffer size increases, such that it is greater than $179sec$ for $B = 200K$.

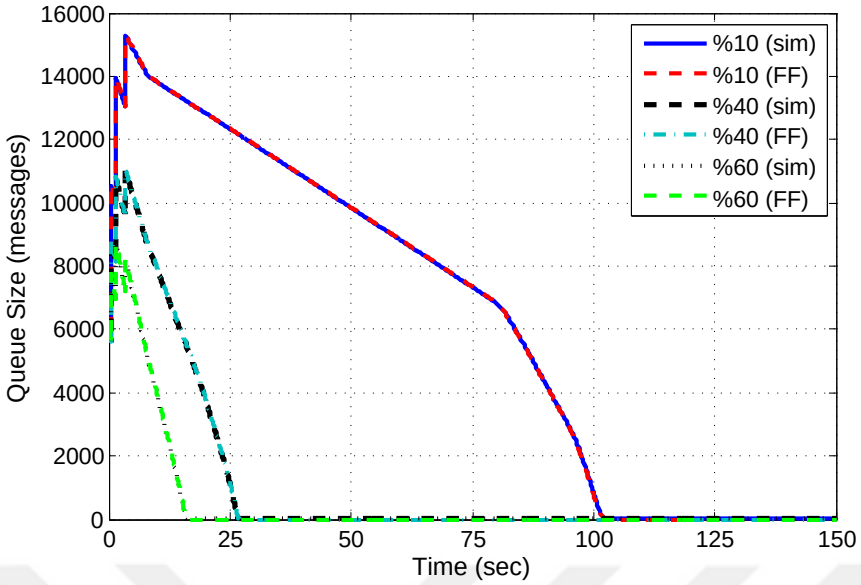


Figure 6.28 : Queue size change for the conventional SIP server when the initial queue size is set to 6K messages, for various loss rates.

6.3 Infinite Buffer Case in Lossy Network Conditions

In this section, we evaluated the performance of the discrete-time fluid-flow models, which are derived for the infinite buffer case under lossy network conditions, for both the conventional SIP and the PRSM, in Section 4.8.1 and Section 4.8.2, respectively. Lossy network conditions may create non-redundant retransmissions on SIP traffic such that these non-redundant retransmissions provide a reliability over the unreliable UDP transport. However, lossy network conditions will likely increase response delays and the number of call establishment failures, especially for the PRSM. We also consider network losses such as the loss rate is varied from 0% to 90% with steps of 10% to evaluate the effects of various network losses.

For all experiments, in order to simulate a certain percentage of message loss in the network including retransmissions, a random number uniformly distributed between 0 and 1 is generated. For example, for simulating a network with 10% message loss rate (or the probability of loosing a message is 0.1), if the value of the random number is equal to or greater than 0.1, the arrival is not considered as a loss, otherwise the arrival is considered as a loss.

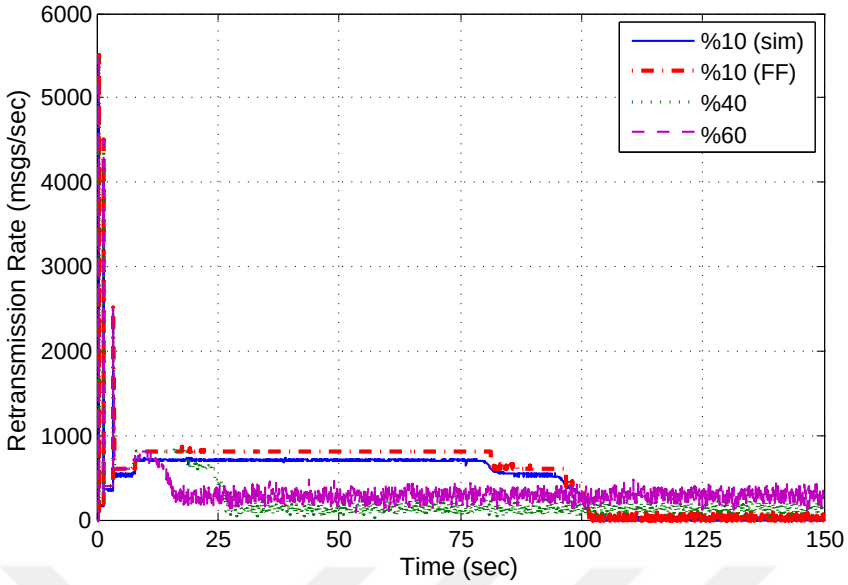


Figure 6.29 : Change of the retransmission rate for the conventional SIP server when the initial queue size is set to 6K messages, for various loss rates.

6.3.1 Overload caused by bulk arrivals

The same test settings in Section 6.1.1 are used in the lossy network case and the test duration is set to 150 seconds. Although original message arrival and service rates are deterministic, because of random characteristics of the loss rate, we repeat the experiments 100 times and report the average value of 100 experiments. Note that there has been no loss in the network when bulk arrivals entered the server at t=0s. So, we assumed that there are no non-redundant retransmissions among these bulk arrivals.

Figure 6.28 shows the queue size behavior of the conventional SIP server under various loss rates using both the analysis tool and the model. Note that we verified the accuracy of the model for other loss rates as we demonstrated in Figure 5.15. Compared to the lossless network case in Section 6.1.1, the network losses cause decreasing the amount of the total offered traffic for the server. Therefore, the conventional SIP can handle the overload condition under the 10% loss rate. Then, the queue size is settled down to almost 0 around $t = 100s$, for the loss rate 10%, since the redundant retransmission rate is settled down to 0 (refer to Figure 6.29), while the non-redundant retransmissions still occur. The amount of non-redundant retransmissions increase when the loss rate increases as depicted in Figure 6.29.

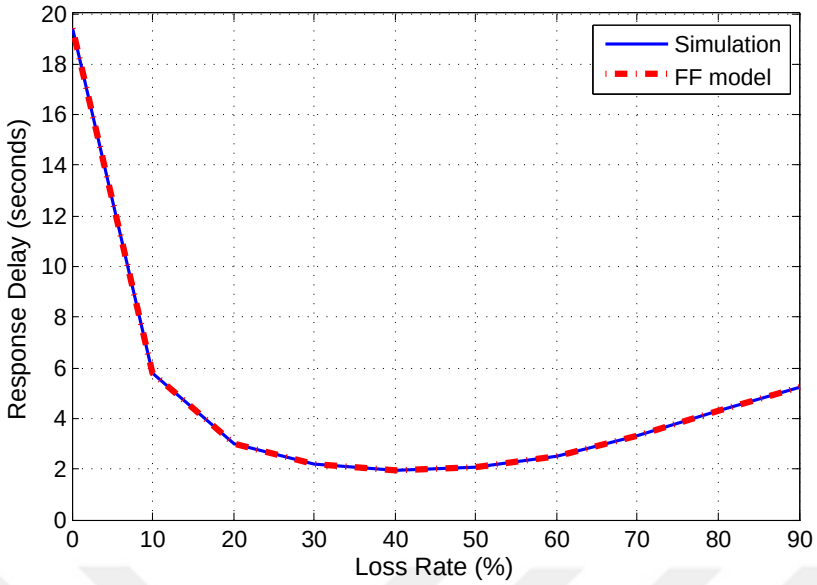


Figure 6.30 : Average response delays for the conventional SIP server when the initial queue size is set to 6K messages, for loss rates between 0%-90%.

The average response delay, as shown in Figure 6.30, decreases from around 19.4 seconds in the lossless network case to around 5.8 seconds when the loss rate is 10%. Similarly, the call failure ratio is decreased from around 0.1 to almost 0 (refer to Figure 6.31). As a positive effect of SIP retransmission mechanism, the call failure ratio remains around 0 up to the 50% loss rate, and then it gradually increases by reaching around 37% when the loss rate is 90%. The average response delay is around 5.7 seconds when the loss rate regime is set to 90%, while it is between 2 and 3 seconds up to the 60% loss rate.

In order to evaluate the accuracy of the model and to obtain an insight about how the model of the conventional SIP server handles original messages and retransmissions, we plot both delay of each individual call in the order of response and the running average [63] (recursive averaging [64]) delay in Figure 6.32 when the loss rate is 10%. These results are obtained from a single execution of the test case with 35.978 calls. Note that the model results match with the results of the analysis tool as we reported in Figure 5.20. The singular peaks in Figure 6.32 indicate higher processing delay since calls are successfully established through retransmitted messages while original message and earlier retransmissions are lost in the network.

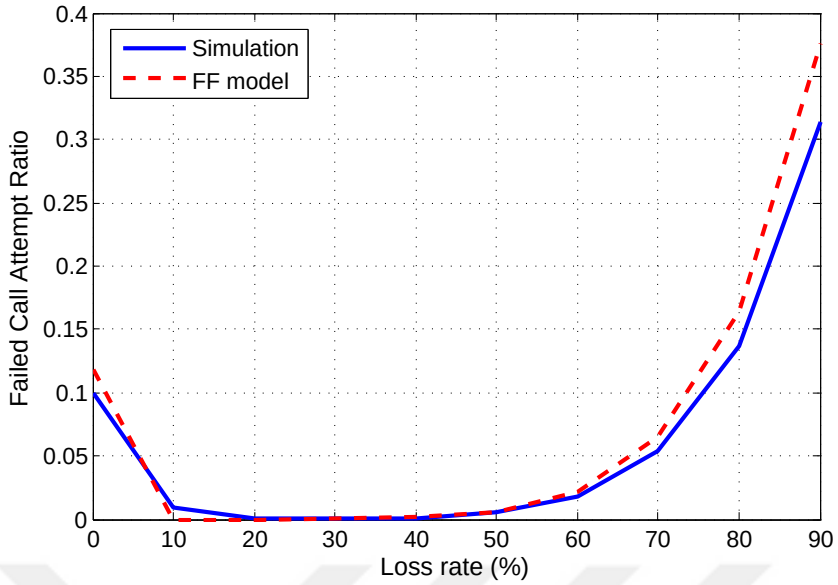


Figure 6.31 : The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the conventional SIP server when the initial queue size is set to 6K messages.

The similarity between the numerical results of the fluid-flow model and the simulation tool demonstrates that the fluid-flow model can accurately capture the behavior of the conventional SIP sever.

Figure 6.33 illustrates the queue size change for the PRSM under various loss rates, as including the result from the analysis tool for the loss rate 10% for the accuracy evaluation. As in the conventional SIP case, the lossy network conditions result in decreasing the offered traffic to the server, as well as retransmission rates (refer to Figure 6.34). Besides, no significant increase on the response delay is observed. As shown in Figure 6.35, the average response delay gradually increases when the loss rate increases, up to the 50% loss rate. For loss rates between 60% and 90%, the behavior of the PRSM is similar to the conventional SIP. As shown in Figure 6.36, there is no significant increase in the call failure ratio for relatively low loss rates, up to 50%. In general, the call failure ratio behaves as similar to the conventional SIP server except the lossless case.

Figure 6.37 shows delays for each call in their order of response and the running average delay for the loss rate 10%. These results are obtained from a single execution of the test case. Note that, we have already provided a similar results from the

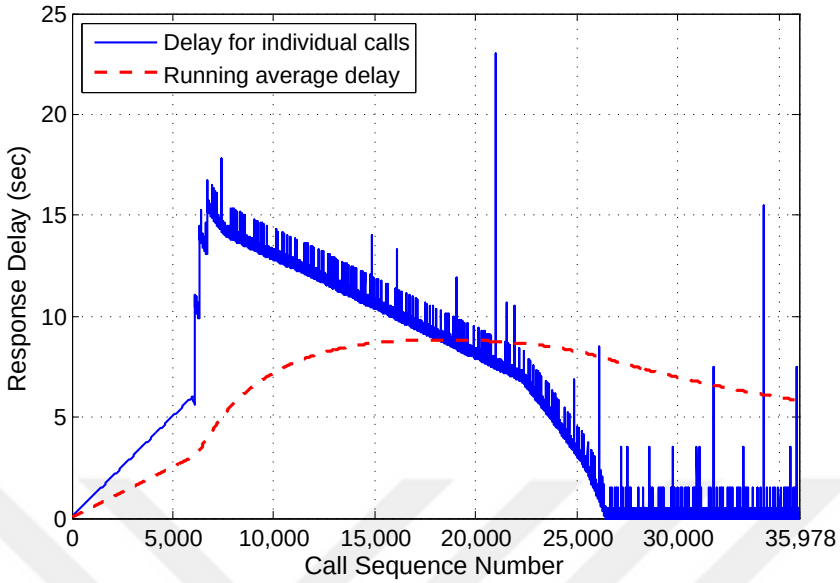


Figure 6.32 : Delay for each individual call in sequence and the running average delay for the conventional SIP under 10% loss rate.

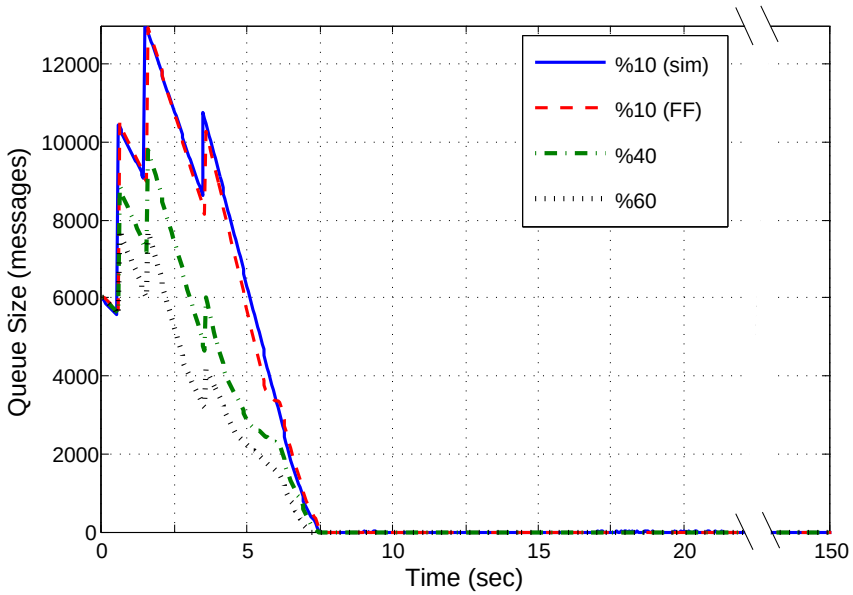


Figure 6.33 : Queue size change for the PRSM when the initial queue size is set to 6K messages, for various loss rates.

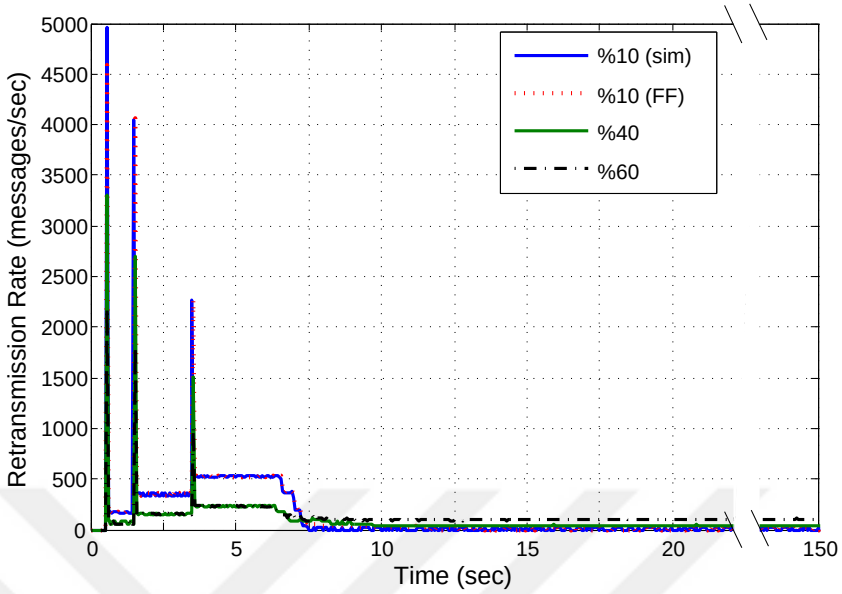


Figure 6.34 : Change of the retransmission rate for the PRSM when the initial queue size is set to 6K messages, for various loss rates.

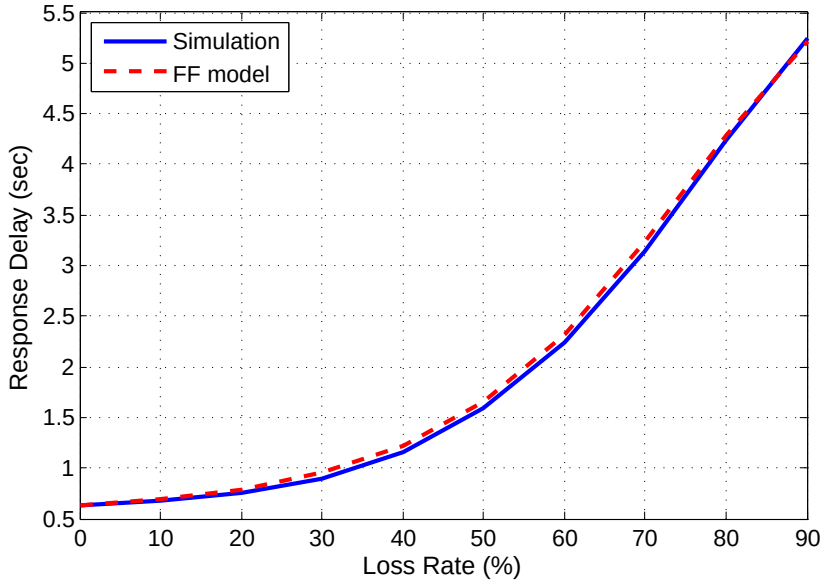


Figure 6.35 : Average response delays for the PRSM when the initial queue size is set to 6K messages, for loss rates between 0%-90%.

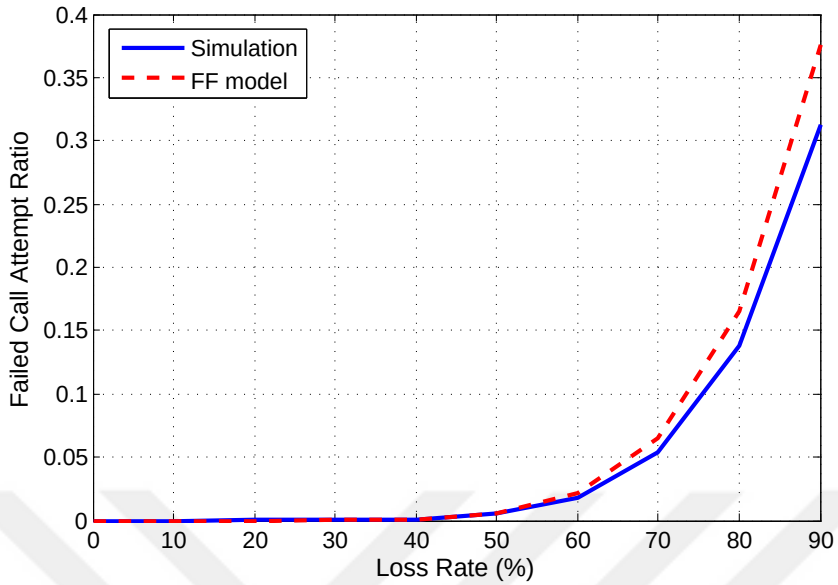


Figure 6.36 : The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the PRSM when the initial queue size is set to 6K messages.

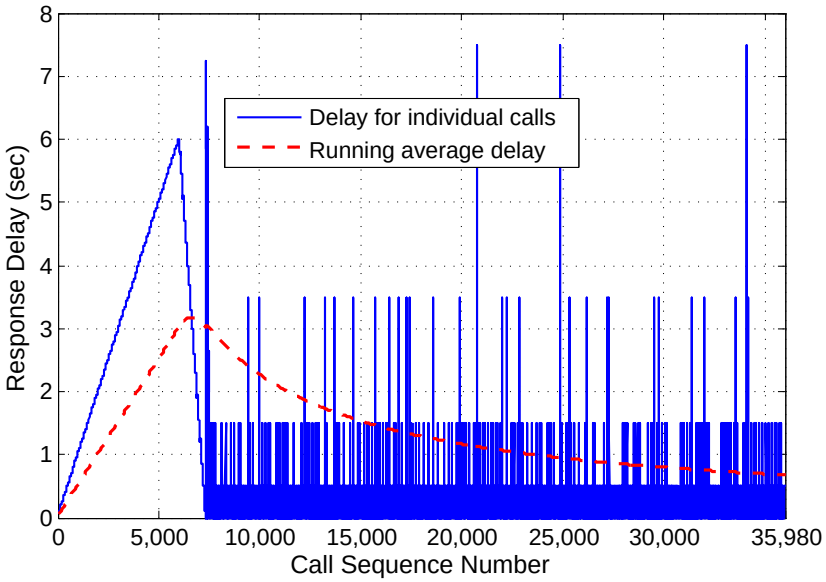


Figure 6.37 : Delay for individual calls in sequence and the running average delay for the PRSM under loss rate 10%.

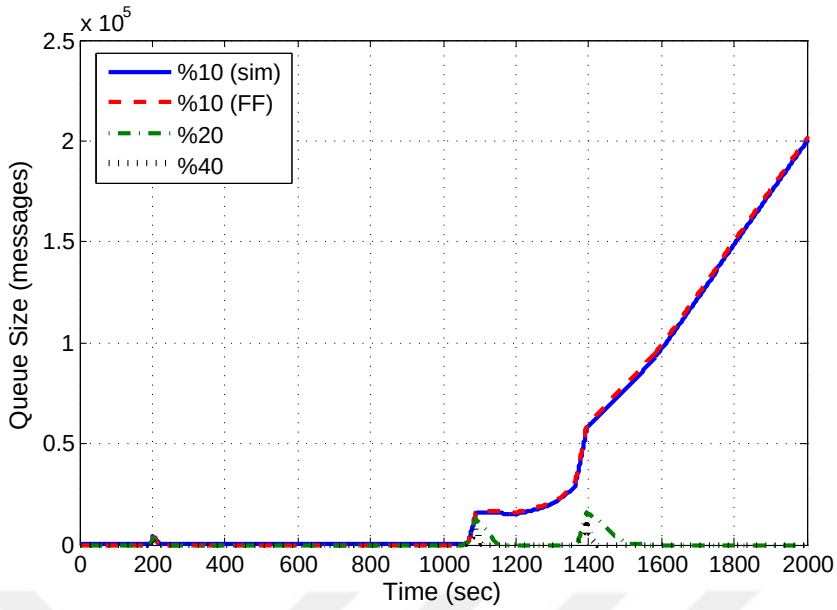


Figure 6.38 : Queue size change for the conventional SIP server for various loss rates when the server is overloaded by temporal slow down.

analysis tool in Figure 5.26. The singular peaks indicate the processing delay of calls that are processed through retransmitted messages, while original message and earlier retransmissions are lost in the network. Note that because of the strict priority scheduling of the PRSM, the server can start to process non-redundant retransmissions after the primary queue is almost emptied after 7325^{th} call with a delay of 7.25 seconds.

We can also say for the PRSM that, the similarity between the numerical results of the fluid-flow model and the simulation tool demonstrates that the fluid-flow model can accurately capture the behavior of the SIP sever with the PRSM.

6.3.2 Overload caused by temporal server slow down

Similar to the settings in Section 6.1.2, in this test case, the SIP server works in one of two alternating modes: normal service or maintenance. Because of the reduced server processing capacity during the maintenance period, the server may be overloaded. The mean arrival rate is $\lambda = 200$ messages/sec; the mean service rate at the normal service mode $\mu_1 = 1000$ messages/sec; the mean service rate at the maintenance mode is $\mu_2 = 180$ messages/sec; all are Poisson distributed. The simulation period is 2000 seconds. We repeated the experiments 100 times and the numerical results are reported as the average value of 100 experiments for each model.

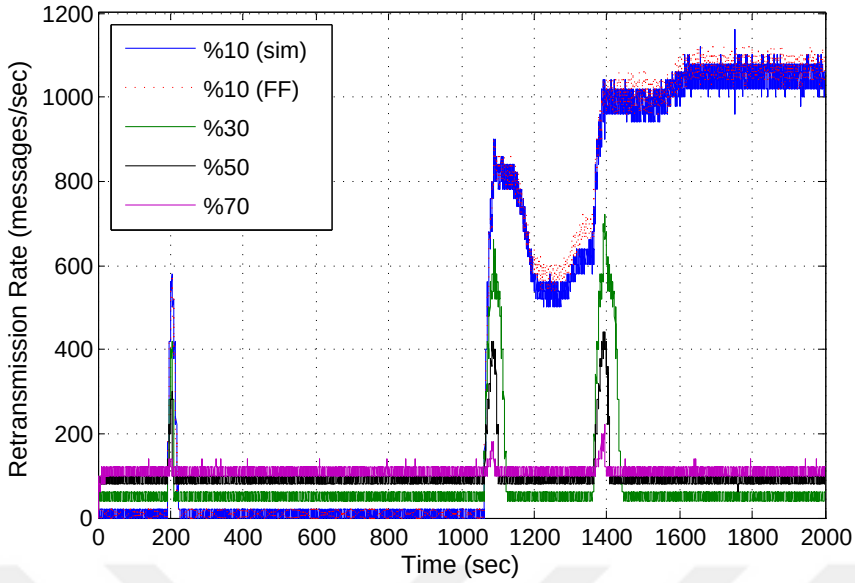


Figure 6.39 : Change of the retransmission rate for the conventional SIP server for various loss rates when the server is overloaded by temporal slow down.

Figure 6.38 shows the queue size of the conventional SIP server under various loss rates. To verify the accuracy of the fluid-flow based model, the figure includes the result of the analysis tool for the 10% loss rate. Although the offered traffic is reduced in the lossy network case, the conventional SIP server still cannot handle the overload condition for the 10% loss rate as demonstrated by the queue size which tends to increase infinitely. The retransmission rate exceeds 1000 messages/sec almost after t=1400s (refer to Figure 6.39). For relatively high loss rates, because a small fraction of the traffic can arrive the server, queue sizes are too small when compared with of 10%. Therefore we do not include results of higher loss rates in Figure 6.38. It can be seen from Figure 6.39 that higher loss rates cause only a background retransmission traffic whose rate increases when the loss rate increases, such that the average retransmission rates are around 50, 90 and 110 for the loss rates of 30%, 50% and 70%, respectively.

Figure 6.40 and 6.41 illustrate the average response delay and the ratio of the number of failed call attempts to the number of all call attempts, respectively, for loss rates between 0%-90%. The average response delay is 29.23 seconds for the 10% loss rate while it is 1.24 seconds for 20% and increases up to 6.67 seconds for 90%. The call failure ratio is reported around 0.28 for the loss rate 10%. The call failure ratio is relatively very low for the loss rates between 20% and 50% and exponentially increases

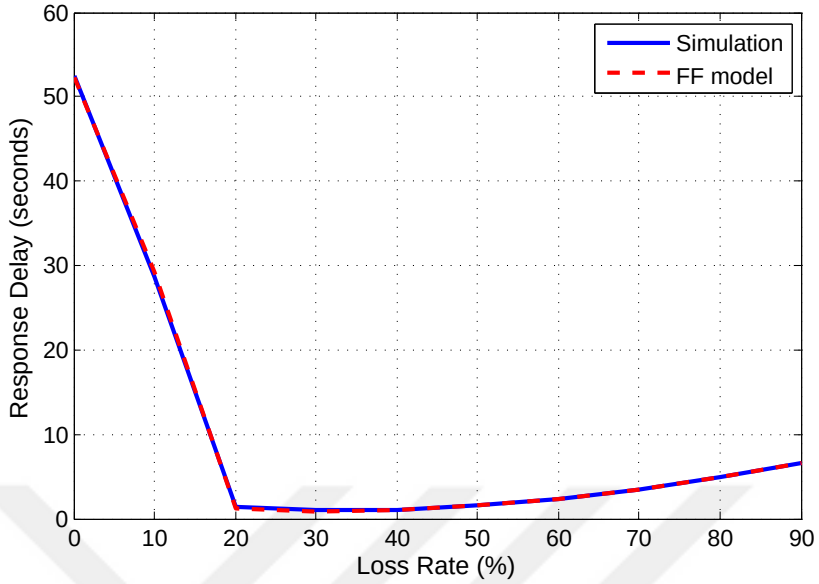


Figure 6.40 : Average response delays for the conventional SIP server for loss rates between 0%-90%.

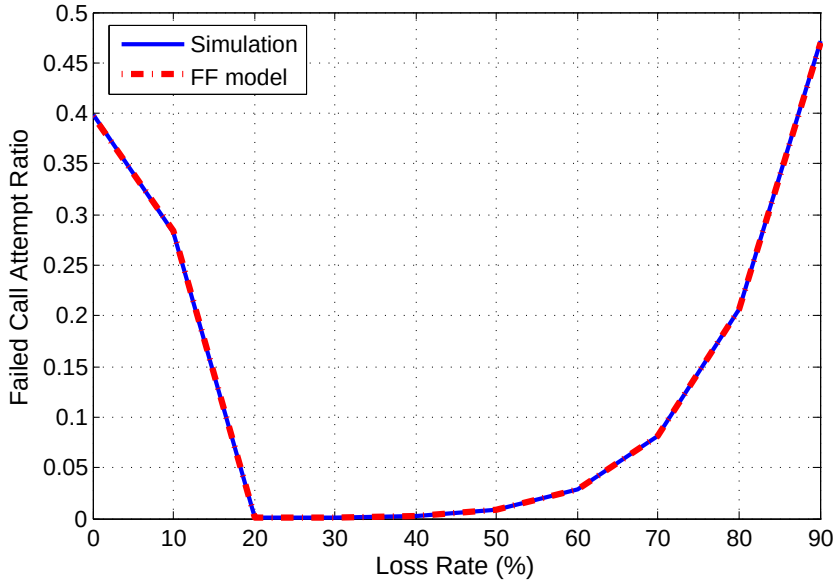


Figure 6.41 : The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the conventional SIP when the server is overloaded by temporal slow down.

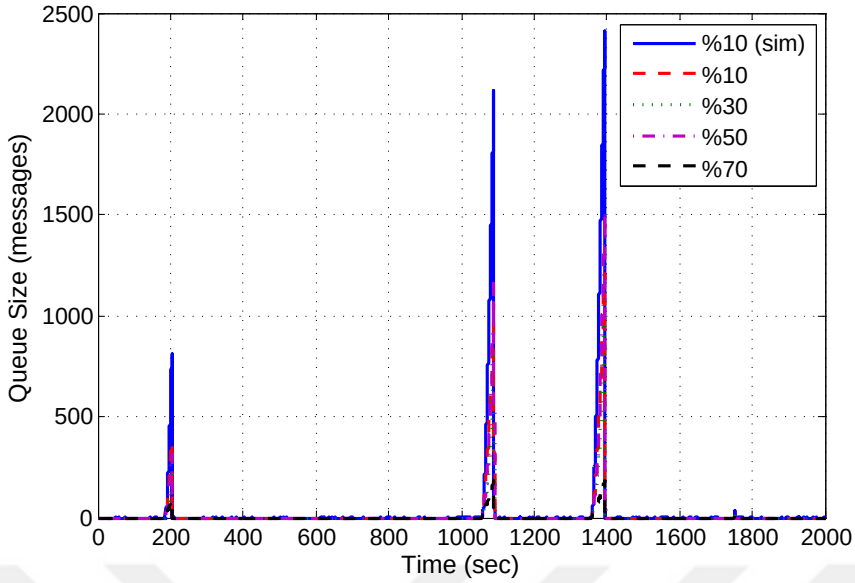


Figure 6.42 : Queue size change for the PRSM when the server is overloaded by temporal slow down, for various loss rates.

as the loss rate increases beyond 50%. It is reported as high as 0.47 for the loss rate 90% since this loss rate represents an extremely lossy network condition.

Figure 6.42 and Figure 6.43 show queue size and retransmission rate changes under various loss rates for the PRSM when the server is overloaded by temporal slow down, respectively. In order to evaluate the accuracy of the model, we report the queue size of the analysis tool when the loss rate is 10% which creates higher queue size compared to the lossless network case illustrated in Figure 6.10. This is because of non-redundant retransmissions that cannot be removed by the removal process. However, this situation does not affect the server with PRSM to handle the overload conditions, since the retransmission rate does not exceed 160 messages/sec as shown in Figure 6.43(a). Higher loss rates create a background retransmission traffic which increases when the loss rate increases, but it does not exceed 180 messages/sec.

As shown in Figure 6.44, for the PRSM, the average response delay do not exceed 1 seconds up to the 50% loss rate. For higher loss rates, the behavior of the PRSM is similar to the conventional SIP server. As shown in Figure 6.45, the average failed call ratio also behaves similar to the previous test case.

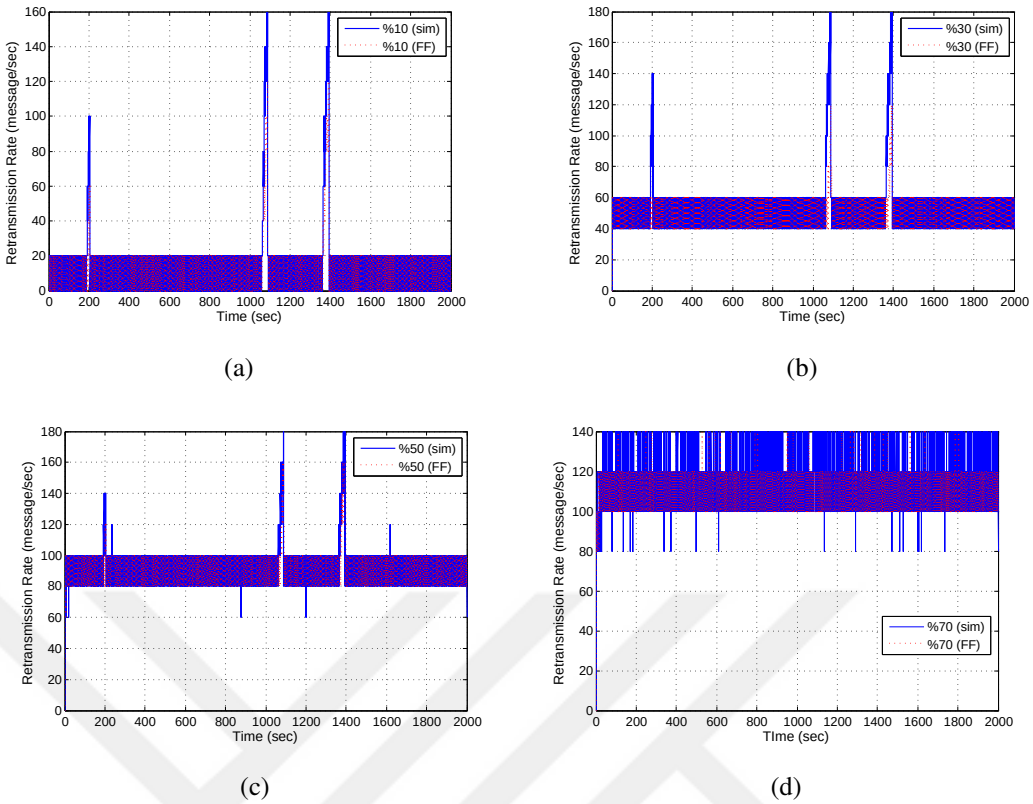


Figure 6.43 : Change of the retransmission rate for the PRSM when the server is overloaded by temporal slow down for loss rates (a) 10%, (b) 30%, (c) 50% and (d) 70%.

The similarity between the numerical results of the fluid-flow model and the simulation tool demonstrates that, the fluid-flow models for both the conventional SIP and PRSM can accurately capture the corresponding server behaviors.

6.3.3 Overload caused by bursty arrivals

By using the parameter settings in Section 6.1.3, this test case employs MMPP traffic model to simulate heavy traffic conditions overloading SIP servers. Note that using this traffic model and under the consideration that there is no loss in the network, we observed that the analysis tool failed for the conventional SIP server simulation because lack of computing resources for very long queue sizes due to the excessive number of retransmissions. We observed that the analysis tool failed for loss rate settings up to 60%, and for higher loss rate settings, it unexpectedly runs too long for even if the loss rate is set to 90%. For this reason, we can report only the results from the models in this section.

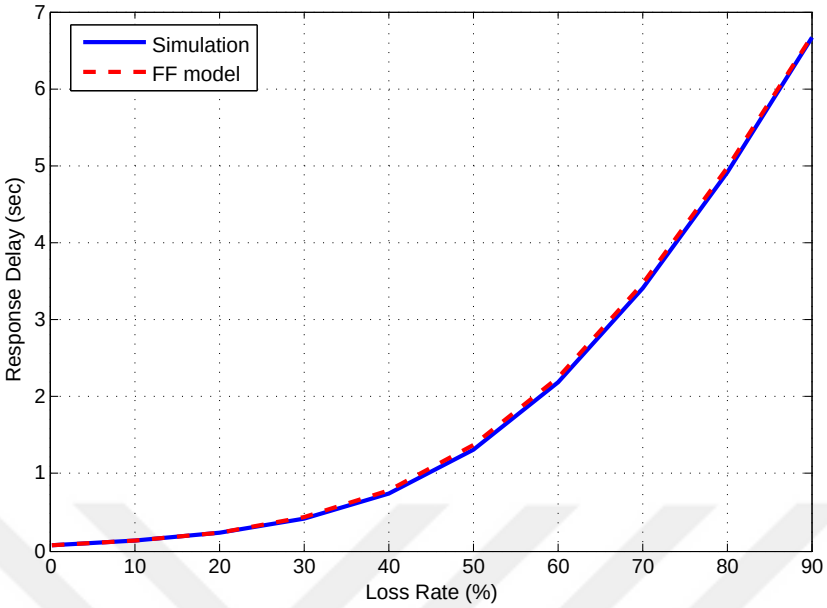


Figure 6.44 : Average response delays for the conventional SIP server for loss rates between 0%-90%.

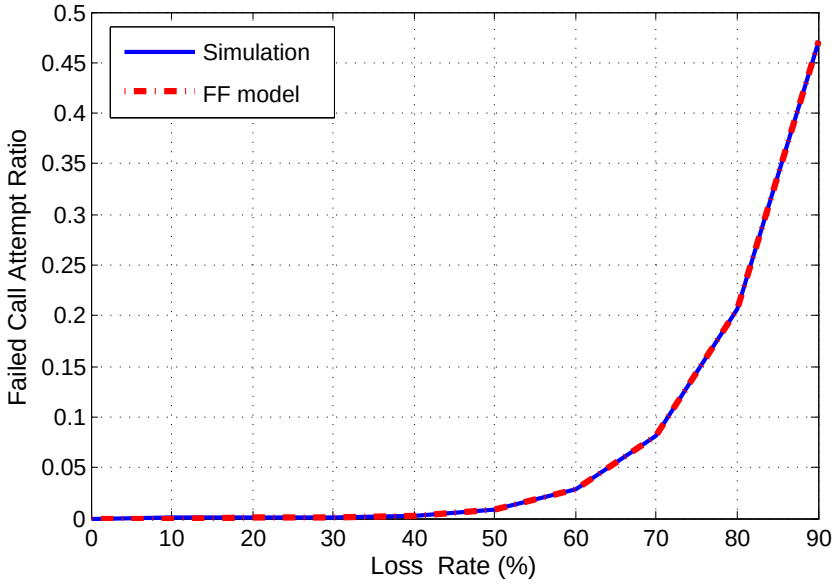


Figure 6.45 : The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the PRSM when the server is overloaded by temporal slow down.

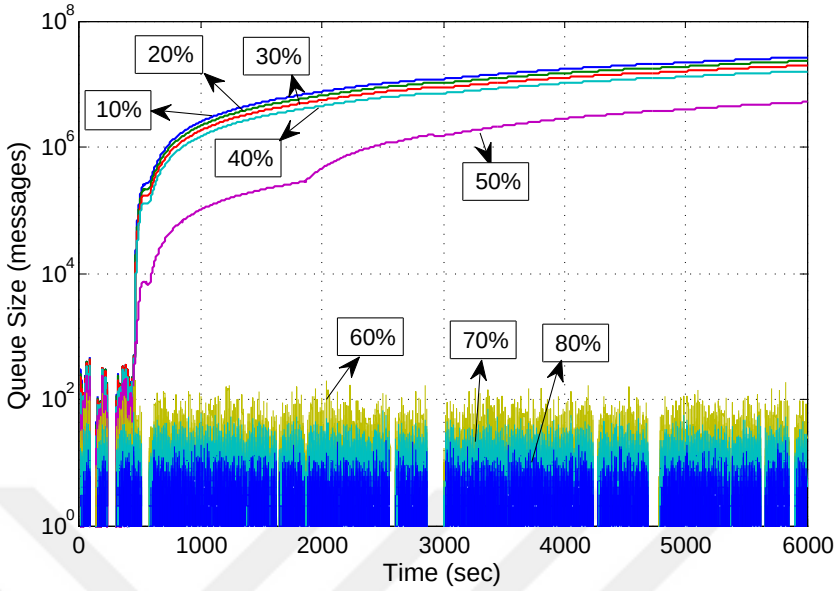


Figure 6.46 : Queue size change for the conventional SIP server under bursty traffic conditions, for various loss rates.

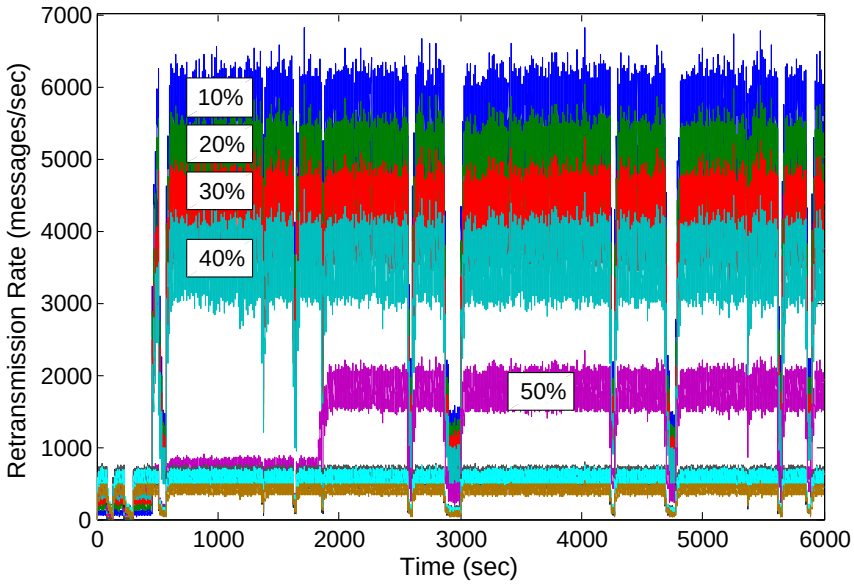


Figure 6.47 : Change of the retransmission rate for the conventional SIP server under bursty traffic conditions, for various loss rates.

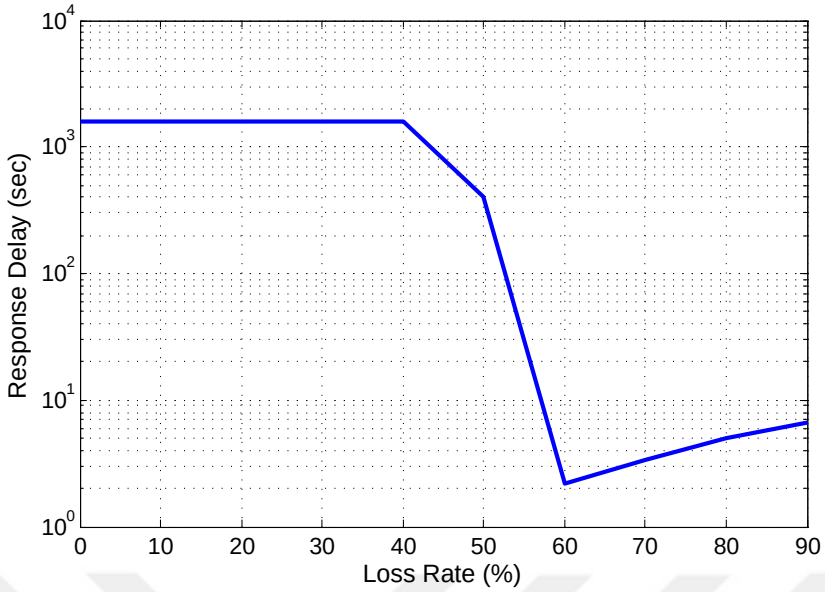


Figure 6.48 : Average response delays for the conventional SIP server for loss rates between 0%-90%.

Figure 6.46 shows the change of queue sizes in time for the conventional SIP server under various loss rates. Up to the 60% loss rate, the queue sizes gradually increase tending to go towards infinity because of higher retransmission rates exceeding 1000 messages/sec (refer to Figure 6.47). It is expected that this situation eventually leads a server crash.

Both the average response delay and the call failure ratio results reflect severe overload conditions up to the 60% loss rate (refer to Figure 6.48 and Figure 6.49, respectively). While the average response delay is around 1600 seconds for loss rates up to 50%, it decreases to around 400 seconds for 50% and 2.17 seconds for 60%. Then, it increases again for higher loss rates, such that it is 6.69 seconds for the 90% loss rate. Similarly, the average call failure ratio is around 0.92 up to loss rates 50%, then it decreases to 0.39 for the 50% loss rate and to 0.03 for 60%. Then, it increases to 0.48 for the 90% loss rate.

For the PRSM, because both the queue size and retransmission rate change versus time are too small, they are not illustrated here. We can note that, the average value of total queue sizes for loss rates between 10% and 50% have higher values than lossless case, such that it is 106.85, which is the highest average value of all loss rates, for the loss rate 30%, while 25.04 for the lossless case. For all loss rates, the average value of the

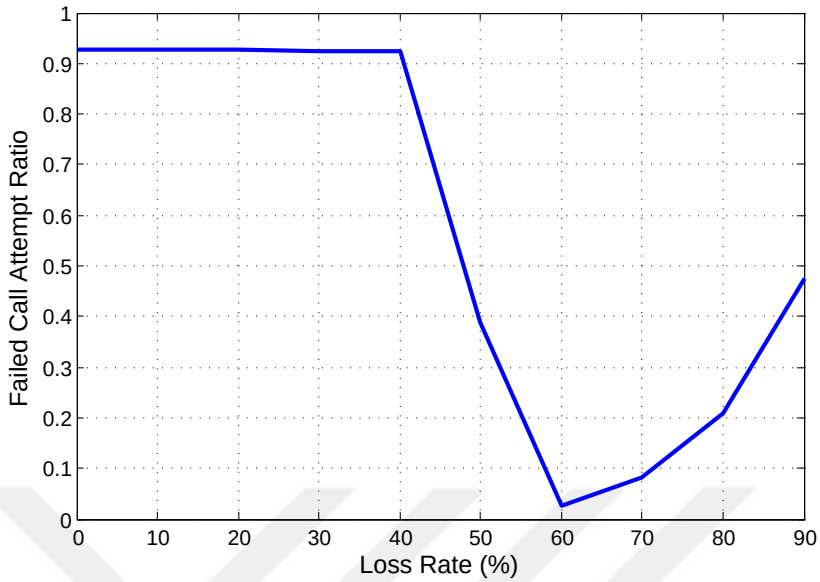


Figure 6.49 : The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the conventional SIP server.

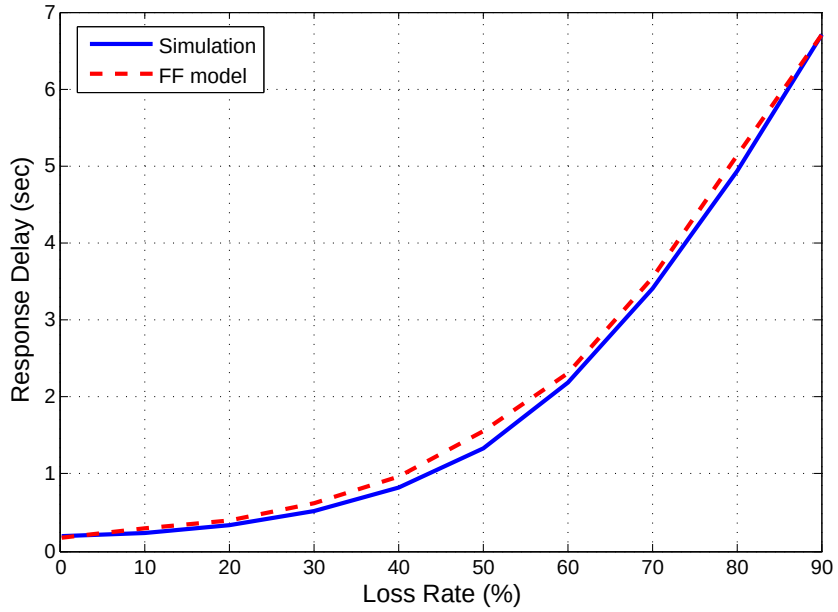


Figure 6.50 : Average response delays for the PRSM server for loss rates between 0%-90%.

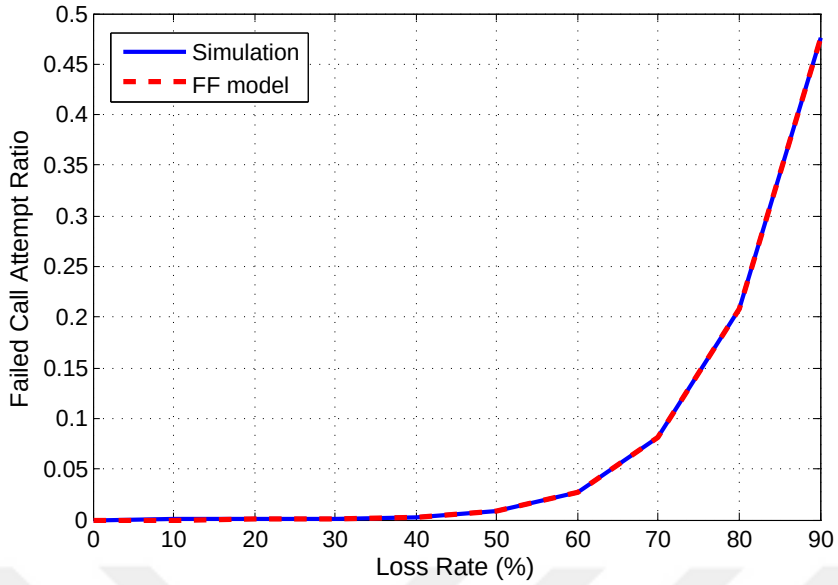


Figure 6.51 : The average ratio of the number of failed call attempts to the number of all call attempts for loss rates between 0%-90% for the PRSM.

secondary is higher than the average value of the primary queue, such that the highest value of the primary queue is 3.98, which is for the lossless case.

As shown in Figure 6.50 and Figure 6.51 the average response delay and the call failure ratio versus loss rates between 0%-90% are similar to those given in the test case for temporary slow down, respectively.

7. A REAL-TIME IMPLEMENTATION WITH JAIN-SIP STACK

We used JAIN-SIP stack [67], which is written in Java, to implement the PRSM as a proof-of-concept to demonstrate that the implementation of the PRSM is feasible, requires minimal changes at the SIP server software and coexists with the conventional SIP server implementation to support backward compatibility. We also have a basic traffic test with SIPp traffic generator [68] to evaluate the implementation under the real-time traffic. Our first observations indicate that the performance of the implementation may be improved by tuning the garbage collection of Java Virtual Machine (JVM) or modifying the software for a garbage collection friendly implementation.

To have a complete test environment, we implemented the server side application in the JAIN-SIP stack according to the requirements of the PRSM as well as the client side application sending retransmission information in the request-line of a request message. To be able to utilize SIPp traffic generator for the PRSM experiments, we also modified the SIPp software to provide retransmission information in the request line of SIP requests. SIPp and JAIN-SIP stack are used together to emulate the scenario of many SIP clients are requesting a service from a single SIP server. As opposed to the regular call scenario, the modified SIP server emulates the behavior of many terminating SIP clients.

An application can configure the JAIN-SIP stack using Java `Properties` class for system parameters such as listening IP address and the port of the server as well as for behavioral parameters such as specifying whether dialog handling will be in the application or stack side. `Properties` include configuration values managed as key/value pairs, such that, both parameters in each pair are `String`. For both client and server implementation, we defined new properties to make the PRSM behavior is configurable. In this way, we can use the same software for both the conventional SIP and PRSM by only changing configuration values.

This chapter is organized as follows: Section 7.1 consists of brief background of JAIN-SIP software architecture. Section 7.2 and Section 7.3 provide the details of the server side and the client side implementation of the PRSM, respectively.

7.1 JAIN-SIP Background

There are three versions of the JAIN-SIP stack [69]. The first version (1.0) was based on the old SIP RFC 2543 . The newer version (1.1) supports the current SIP RFC 3261. The version 1.2 consists of some enhancements to the version 1.1. We used the version 1.2 to implement the PRSM.

The JAIN-SIP stack provides the standardized Java interface to the SIP for client and server applications. The implementation manages processing and life-cycles of the requests, responses, transactions, dialogs and the mappings between them. Figure 7.1 illustrates the high level software architecture of JAIN-SIP Stack. An

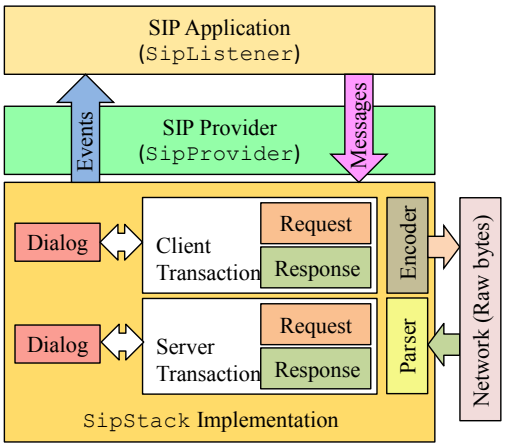


Figure 7.1 : The Architecture of JAIN-SIP Stack Implementation.

application implements `SipListener` interface to interact with the `SipStack` and registers with `SipProvider` for all messaging capabilities of the stack. `SipProvider` allows applications to send and receive SIP messages. Applications receive messages from the stack as encapsulated into `Event` via the `SipListener` interface. Event mechanism is unidirectional, applications can send messages to the stack by passing request and response messages through `SipProvider`. The event processing mechanism is defined by an event-listener interface that includes a "`processEvent`" operation for each type of event. When the SIP stack receives

request and response messages from the network, SipProvider passes them as events on to the event listener, SipListener.

The SipStack interface defines the methods to control the architecture and setup of the SIP stack. A single SipStack has to be created for each IP interfaces, so there is a one-to-many relationship between a SipStack and a SipProvider. The SipStack interface allows applications to manage and configure the underlying SIP stack. This configuration is specified through Java Properties class. The application can start or stop the SIP stack. The SipStack interface also contains operations to create instances of SipProvider and ListeningPoint.

An instance of ListeningPoint class is a Java representation of the socket. The ListeningPoint interface allows the application to set transport parameters IP, port and transport type. The transport type may be set as TCP, UDP or SCTP [1]. SIPStack may use several sockets for communicating with the network. Therefore, there is a one-to-many relationship between a SipStack and a ListeningPoint.

The creation of an instance of ListeningPoint class entails the creation an instance of MessageProcessor interface as depending on the selected transport for this specific ListeningPoint. There are two implementations of MessageProcessor, one for connection-oriented protocols like TCP and SCTP, ConnectionOrientedMessageProcessor and one for UDP, UDPMessageProcessor. The UDPMessageProcessor owns an instance of DatagramSocket of Java to get messages from the network. It also owns a BlockingQueue which is instantiated by LinkedBlockingQueue to queue the received messages. The queue is processed by UDPChannels which are bound in a group of Java thread blocked in the queue. The number of threads are configurable by the application. At runtime, the UDPMessageProcessor is blocked on the socket and waits for messages, while UDPChannel threads are blocked on the queue and wait for messages. Whenever a message is received from the network, an instance of DatagramQueuedMessageDispatch class is created to encapsulate the received datagram message, and put into the BlockingQueue. Then a UDPChannel thread receives the message from the queue to process. The processing of a received message consists of the parsing

[1] To support SCTP, Java version 7 (or higher) is required

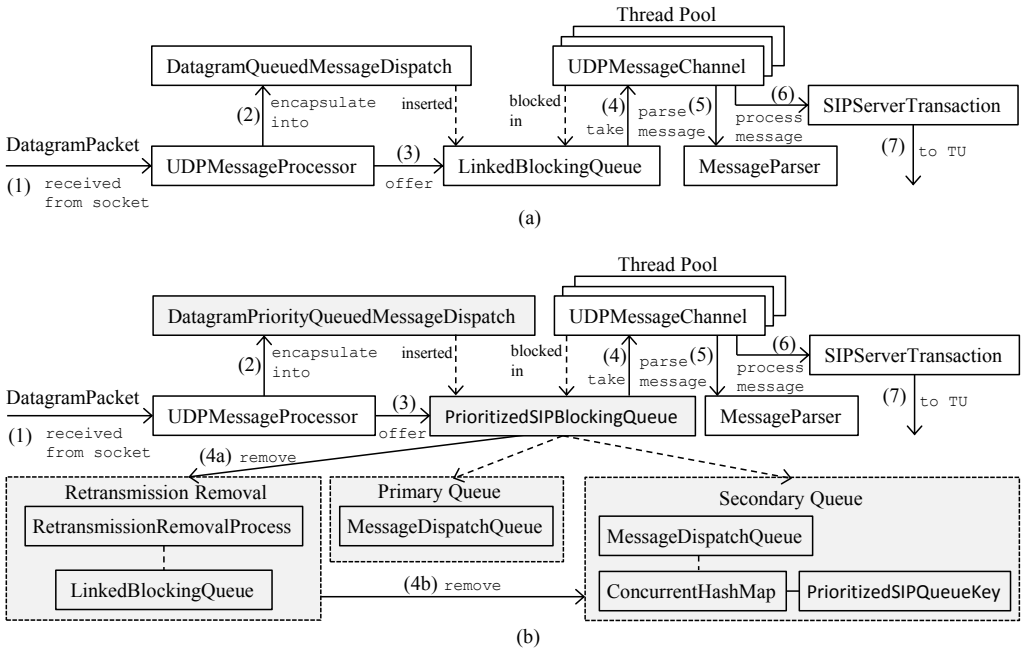


Figure 7.2 : (a) High level functional view of a typical SIP Implementation up to Transaction Layer,
(b) Its extension with Request Prioritization Mechanism).

of the message by using `MessageParser` and calling `processRequest()` if it is a request message or `processResponse` if it is a response message, through `ServerRequestInterface` or `ServerResponseInterface`, respectively. The `ServerRequestInterface` interface is implemented by `SIPServerTransaction` while the `ServerResponseInterface` interface is implemented by `SIPClientTransaction` classes. Then, after processing through the transaction state machine, the message reaches to the application, which is an implementor of TU.

7.2 SIP Server Implementation with PRSM

We modified its UDP message handling mechanism to implement our priority based message queuing approach. The high level UDP message processing of the existing JAIN-SIP stack is illustrated at Figure 7.2-(a). The `UDPMessageProcessor` is the listener for UDP port and the receiver from UDP socket. If the stack is configured for working with multithreads, a UDP datagram packet, received from the socket, is offered to the message queue. The message queue is an instance of Java's `LinkedBlockingQueue` as encapsulated in a `DatagramQueuedMessageDispatch` object instance. A thread-pool formed

by `UDPMessageChannel` object instances is responsible for parsing and further processing through `SIPServerTransaction`.

As illustrated in Figure 7.2-(b), our implementation is based on `PrioritizedSIPBlockingQueue`, where it implements non-preemptive priority queuing scheme (i.e., a message in service is not interrupted by new incoming arrivals even if they have higher priority). Messages which have the same priority are queued in a FIFO (First-In-First-Out) manner. For the current implementation, we assumed only two priority levels; *"priority-0"* and *"priority-1"*. Assuming new arrival requests take precedence over retransmissions, the former is for new arrival requests, and the latter is for retransmitted requests. Any precedence within the retransmitted requests is not assumed. The Retransmission Removal is implemented as a configurable facility. If activated, when a request is polled from the queue for processing, all corresponding retransmissions are removed from the queue.

Similar to `LinkedBlockingQueue`, `PrioritizedSIPBlockingQueue` is an optionally bounded blocking queue that blocks when a thread attempts to de-queue from it and the queue is empty, until some other thread inserts an item into the queue. Actually, it is a wrapper for two inner prioritized queues named *"primary"* and *"secondary"*, for *"priority-0"* messages and for *"priority-1"* messages respectively. For example for a de-queue request, it checks the primary queue first for available queue element, if it is empty, then it checks the secondary queue; if both are empty, the caller is blocked. The primary queue is an instance of `MessageDispatchQueue` which implements a doubly linked queue software component. The secondary queue consists of an instance of `MessageDispatchQueue`, and a hash-map container, Java's `ConcurrentHashMap`, for fast access to retransmitted messages on "retransmission removal procedure". The key, `PrioritizedSIPQueueKey`, for the hash-map is obtained from *"rseq-num"* and *"r-branch"* values of *"rb"* parameter (refer to Section 3.2.1). The doubly-linked structure of the queue provides ability for efficient removal.

The buffer size of `PrioritizedSIPBlockingQueue` is configurable. In the case of a limited buffer size assigned for the queue, the buffer size limitation is applied to the total of both primary and secondary queues, e.g. for buffer size is 100, there can be 100 requests, in total, in the message queue which cover all inner queues.

The buffer size is $2^{31} - 1$ by default which is defined by the implementation of `LinkedBlockingQueue`.

`RetransmissionRemovalProcedure` is triggered with polling a request message from either the primary or the secondary queue for processing. To invoke the procedure, `PrioritizedSIPBlockingQueue` replaces the job including *"rseq-num"* and *"r-branch"* into `LinkedBlockingQueue` of the task of the retransmission removal procedure.

`DatagramPriorityQueuedMessageDispatch` is the subclass of `DatagramQueuedMessageDispatch` and implements *"request pre-parsing"* to determine the priority of received requests. When it is created to encapsulate a SIP message, it parses the first line to determine whether it is a request message. If it is a request message, it parses request-line to obtain retransmission sequence number and transaction identifier information from *"rb"* parameter. Then, it provides requested priority and identifier for the encapsulated message during the priority queuing. It returns *"priority-0"* for response messages and request messages without *"rb"* parameter, as well as request messages with retransmission sequence number is equals to zero. For requests having retransmission sequence number is greater than zero, it returns *"priority-1"*.

The class `RequestLine` is modified for parsing retransmission information from the SIP request-line. To be able to process *"rb"* parameters, two new class attributes are added into the class; an integer attribute for retransmission sequence number, *"rseq-num"*, and a *String* attribute for *"r-branch"* parameters, with default values of "-1" and "null"-`String`, respectively. The parsing algorithm is modified as based on the number of the tokens in the request-line, that are separated by spaces. In the case of three tokens, it assumes a request-line belongs a conventional SIP request message, and parsing continues in this way, such that *"rb"* related attributes will have default values. In the case of four tokens, it assumes a request-line consisting of the retransmission information on the third token, and parses the token to extract *"rseq-num"* and *"r-branch"* values. If the priority based message queuing mechanism is activated in the system, it first gets the value of the attribute for *"rseq-num"*. If it is "-1", the mechanism assumes that this request message does not contain retransmission information and considers as a "priority-0" message (however, note that, when it is

pooled from the queue for processing, no removal procedure will be applied for this message), otherwise it put into the corresponding priority queue according to the value of the parameter. This parsing and processing algorithms enables the implementation to handle both type of request messages, with and with no *"rb"* parameters in the request-line, at the same time.

At TU (Transaction User) layer, we implemented a simple structure as based on samples provided by JAIN-SIP to handle a basic call setup. As recommended at RFC 3261, JAIN-SIP INVITE server transaction starts a timer with 200 ms to generate 100-Trying response if TU cannot generate a provisional or final response within that time. Therefore, TU implementation generates 180-Ringing response upon receiving an INVITE. Then it creates a `TimerTask` for 1 second to send 200-OK. This provides a multi-threaded structure to handle multiple calls.

During the implementation, all the existing message processing structure is kept as it is and switching between existing structure and priority based message queuing mechanism is controlled with configuration parameters. As following the configuration approach used of JAIN-SIP which is based on Java system properties, we defined three new system properties are to invoke priority based message queuing mechanism:

(1) `gov.nist.javax.sip.PRIORITY_QUEUING`, which takes values *"true"* and *"false"*. It activates the priority based queuing, if it is set to *"true"*. Default value is *"false"*.

(2) `gov.nist.javax.sip.REMOVE_RETRANS` which takes values "true" and "false". The retransmission removal procedure of the PRSM is activated if it is set to *"true"*, if priority queuing is activated. Default value is *"false"*.

(3) `gov.nist.javax.sip.APPLY_BUFFER_SIZE`; an integer value greater than 0. -1 is for pseudo-infinite (i.e. $2^{31}-1$)). Determines the total queue size of the priority queues. The default value is $2^{31}-1$).

7.3 SIP Client Implementation Sending Retransmission Information

The modified `RequestLine` class to include additional attributes for *"rseq-num"* and *"r-branch"* parameters, is also used in the client-side implementation for

encoding the request-line into messages to be sent. As stated in Section 7.2, the default values of these attributes are "-1" and "null"-`String`, respectively. `SipProviderImpl.createClientTransaction()` method is modified to set the created branch-id into the attribute for *"r-branch"* parameter of `RequestLine`. This method is invoked whenever `PriorityClient` invokes `getNewClientTransaction()` method from the instance of `SipProviderImpl`.

Whenever the client application supporting the retransmission information in the request-line, creates a request message to be sent, it needs to set *"rseq-num"* attribute to "0". It may also calls `setNotifyOnRetransmit()` method from `ClientTransactionExt` to get timeout notification on each request retransmission. `SIPClientTransactionImpl` sets *"rseq-num"* attribute of `RequestLine` for each retransmission timer expiry as increased by one to be set into the retransmitted request. `RequestLine` uses these set values during the encoding SIP request-line into the transport layer when the request sent to the remote server. For inserting the retransmission information into the request-line, `RequestLine.encode()` method controls the value of *"rseq-num"* attribute. If it equals or is greater than zero, it inserts the retransmission information into the message to be transported. Otherwise, no retransmission information is included in the request-line of the message. This provides the backward compatibility for clients that do not encode retransmission information. In this way, a client application can include the retransmission information in the request messages without a need for configuration parameters.

7.4 Traffic Performance Tests

In this section, we evaluate the performance of JAIN-SIP implementation of the PRSM under real-time traffic provided by SIPp traffic generator following given modifications on SIPp software to provide retransmission information in the request-line of INVITE request messages as defined in Section 3.2.1.

```
<send retrans="500">
   <![CDATA[
      INVITE sip:[service]@[remote_ip]:[remote_port] rb=[nretrans]:[branch] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 1 INVITE
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]
```

Figure 7.3 : An example of INVITE message in a SIPp scenario file as including "rb" parameters.

7.4.1 Modifications on SIPp software

SIPp is an open source test and traffic generator for the SIP protocol. It uses XML (eXtensible Markup Language) based scenario files defining SIP call flows to be tested. It provides statistics about running tests (call rate, round trip delay, and message statistics) in CSV (Comma-Separated Values) format. It supports TCP and UDP over multiple sockets or multiplexed with retransmission management and dynamically adjustable call rates. For some special actions in scenarios and variable parameters in a SIP message can be defined by keywords.

We modified version 3.3.990 SIPp software to implement the behavior requires for the PRSM. We defined "`[nretrans]`" keyword and reused "`[branch]`" keyword to include *"rb"* parameters in request-lines. The `message` class is modified for definition and handling of this keyword. Figure 7.3 illustrates an example of in a SIPp scenario with a INVITE message including retransmission parameters.

For retransmitting requests, SIPp implementation uses stored byte-stream representation of messages from the first built, i.e. the message is not re-built from the scenario representation again. Therefore, in order to increase the sequence number of retransmissions, we saved the position of the [nretrans] keyword in the byte-stream, and modified the value in the case of retransmissions. The `createSendingMessage` and `run` methods of the `call` class is modified for the first insertion and retransmit modifications of the [nretrans] keyword, respectively. For *"r-branch"* parameter, the value created for SIPp [branch] keyword is re-used.

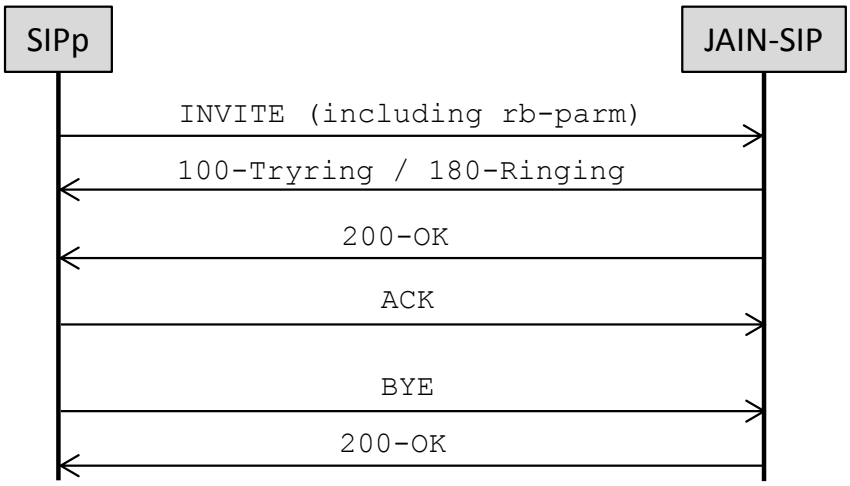


Figure 7.4 : The call scenario set for traffic tests.

7.4.2 Traffic test setup and results

Figure 7.4 illustrates the call flow used for performance tests. For the client side, the SIPp software is extended to include *"rseq-num"* and *"r-branch"* in request-line of each SIP request message. SIPp and JAIN-SIP stack are used together to emulate the scenario of many SIP clients are requesting a service from a single SIP server. As opposed to the regular call scenario, the modified SIP server emulates the behavior of many terminating SIP clients. Both computers running the modified JAIN-SIP and SIPp have 4 Intel Xeon 2.40 GHz processors with 4GB RAM and are connected with 100Mbit Ethernet.

The performance results are obtained using the conventional SIP and the PRSM. To create congestion, we set only one thread in the thread-pool of JAIN-SIP stack (by using the configuration property `"gov.nist.javax.sip.THREAD_POOL_SIZE"`). The congestion control mechanism of JAIN-SIP is set to off, with setting `"gov.nist.javax.sip.CONGESTION_CONTROL_TIMEOUT="0"`. A basic call scenario is selected for SIPp so that each SIP client in SIPp sends BYE. The TU layer of JAIN-SIP is configured to answer the incoming call by sending 200-OK within 1 second after transmitting 180-Ringing. In the PRSM case, only INVITE request is set to convey “rb” parameter.

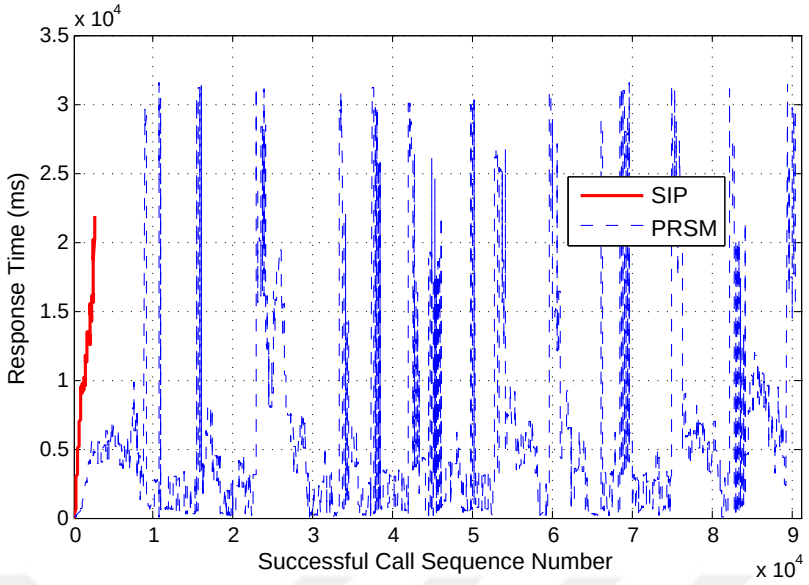


Figure 7.5 : The response delays measured in real-time experiments.

By using the remote control feature of SIPp, we created MMPP like behavior, where the call arrival rate is 100 for 2 seconds and then increases to 995 for 10 seconds; such that this pattern continues for 600 seconds. To make a fair comparison with the original SIP method, the total buffer sizes of both methods should be the same. For all of the real experiments, the buffer size is set to a pseudo infinite value (i.e., limited by the availability of computing resources). The queue implementation mechanism (a single shared queue or two separate queues for low and high priority messages) does not have any effect when the buffer size is not limited.

Figure 7.5 illustrates response times which are measured in SIPp as time difference between sending the first request to the server and receiving a response from the server. We observed that the original SIP without using PRSM crashed around 2600th successful call (at almost 40th second of 600 seconds) for all trials, while our proposed PRSM completed all the tests without crashing. The response time of the original SIP keeps increasing due to the processing time for excessive redundant retransmission messages. PRSM using its priority and removal mechanism keeps redundant retransmissions under control and hence completes this test scenario without causing a server crash. The bursty nature of the traffic results in high response delays whose values exceeding 32 seconds cause call failures. Our measurements indicate that the time required for the message parsing and processing takes about 1693.3

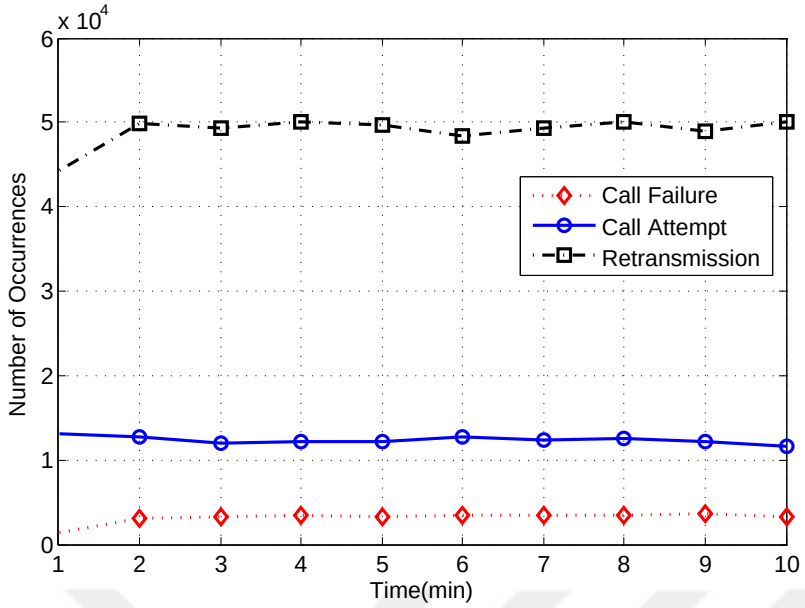


Figure 7.6 : Operational measurements of PRSM reported for every 1 minute.

μs, the time required for the removal process and pre-parsing takes about 18.1 μs and 1.9 μs, respectively. These results verify our assumption that the time required for pre-parsing and removing retransmissions is very small compared to the message parsing and processing. Note that the congestion control mechanism of the JAIN-SIP implementation, which blindly removes both the original and retransmission messages when the queuing delay exceeds 8 seconds, is set to off. Having this blind congestion control mechanism is a violation to the SIP standard and needs to be set off since the SIP standard allows a queuing delay of 32 seconds.

Table 7.1 : Reported mean and standard deviation.

	#Call Attempts		**#Call Failures**		**#Retransmissions**	
	10sec	**1min**	**10sec**	**1min**	**10sec**	**1min**
Mean	2062.3	12373.8	529.2	3175.0	8168.3	49009.6
Std.dev.	382.8	432.8	210.6	634.1	1554.8	1731.3

Figure 7.6 illustrates operational measurements of PRSM reported for every 1 minute interval at SIPp. Each measurement corresponds to the average of 20 experiments. The metric call attempt represents the number of originating calls, the metric call failure represents the number of calls failed due to timeouts, and the metric retransmission represents the number of retransmission of SIP requests. Although the conventional SIP crashes for all these experiments, JAIN-SIP with the PRSM capability copes with a high number of retransmissions and provides a low number of failed calls.

Table 7.1 shows the statistical data such as the average and standard deviation of these metrics. To demonstrate the detailed behavior, the table also includes the data measured for every 10 sec interval. This statistical data indicates that our proposed priority mechanism provides consistent operation by yielding lower standard deviation results. The preliminary results of the JAIN-SIP stack implementation support the results of our analysis tool and fluid-flow models that the PRSM at the SIP server provides consistently better scalability.

8. CONCLUSIONS AND FUTURE WORK

In this thesis, we introduced a new priority-based SIP server request scheduling mechanism (PRSM) which keeps the SIP server queue occupancy under control and hence diminishes the negative effect of redundant retransmissions. The main idea is to identify retransmitted messages before queuing and to provide strict priority for the original incoming SIP requests over the retransmissions, with a minimal message parsing effort. To achieve this objective, the PRSM defines a parameter in the SIP request-line consists of the retransmission sequence number and the request identification. A proof-of-concept implementation of the PRSM by using JAIN-SIP stack indicated that the implementation of the approach is feasible and requires minimal changes at the SIP protocol. The experiments showed that the time required for parsing of the request-line to extract the retransmission information takes 0.11% of the time required for parsing of a request message entirely.

While providing higher priority to original requests enables the server to limit the number of retransmissions by timely processing of messages and providing a response message to the originator, the removal of redundant retransmissions from the system when the corresponding request is processed will consume additional processing power of the server. The preliminary results of the performance tests of the JAIN-SIP implementation using the SIPp traffic generator indicate that the PRSM provides consistently better scalability at the SIP server. We also evaluated the performance of the PRSM using our retransmission analysis tool, which is developed as part of this thesis, and the derivation of discrete-time fluid-flow models for the PRSM.

Performance evaluations with both the analysis tool and models indicate that for the servers with smaller buffer sizes than buffer sizes that causes response delays greater than the first retransmission timer, the conventional SIP server performs better compared to the PRSM. It is an expected result that, queue blocking increases non-redundant retransmission traffic in the network which is the low-priority traffic for the PRSM. However, for higher buffer sizes, the PRSM performs significantly

better than the conventional SIP, because higher buffer size values brings redundant retransmissions due to higher queuing delays. Besides, lower buffer sizes may increase the traffic in the network because of unnecessarily caused non-redundant retransmissions, and in practice this may be unacceptable. The PRSM provides network administrator with the ability to configure the buffer size of a SIP server to a moderately high value as having significantly and consistently better scalability compared to the conventional SIP.

8.1 Future Work

The future work will focus on extending the finite buffer modeling to consider lossy network conditions for both the conventional SIP and the PRSM. This may be a remarkable contribution towards modeling SIP retransmission traffic on wireless networks. From the PRSM perspective, all modeling cases can be studied to derive the case that retransmissions have higher priority over original request messages.

Although enterprises and over-the-top providers can immediately use the PRSM by deploying SIP servers in their networks and providing own SIP clients to the customer, for the more common usage, a standardization of the PRSM is needed, which will be considered as a future work.

During the proof-of-concept implementation of the PRSM, we had problems with Java garbage collection mechanism frequently stopping the message processing from network interfaces. The current design may be enhanced to have a garbage collection friendly implementation. This will provide better real-time environment to evaluate the performance of the PRSM. There is another approach to have a real-time implementation that the PRSM can be implemented on a open-source SIP stack based on native programming languages like C/C++.

Although the current implementation of the analysis and simulation tool may be directly used or easily modified for some other purposes, having a graphical user interface (GUI) may facilitate its usage.

REFERENCES

[1] **Rosenberg, J.**, **Schulzrinne, H.**, **Camarillo, G.**, **Johnston, A.**, **Peterson, J.**, **Sparks, R.**, **Handley, M. and Schooler, E.** (2002). *SIP: Session Initiation Protocol*, RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878, 7462, 7463.

[2] **Rosenberg, J.** (2008). Requirements for Management of Overload in the *Session Initiation Protocol*, RFC 5390, December 2008.

[3] **Hilt, V.**, **Noel, E.**, **Shen, C. and Abdelal, A.** (2011). Design Considerations for *Session Initiation Protocol (SIP)* Overload Control, RFC 6357, August 2011.

[4] **Cortes, M.**, **Ensor, J. and J.O., E.** (2004). On *SIP* performance, *Bell Labs Technical Journal*, *9*(3), 155 – 172.

[5] **Wanke, S.**, **Scharf, S.**, **Kiesel, S. and Wahl, S.** (2007). Measurement of the *SIP* parsing performance in the *SIP* Express Router, *13th Open European Summer School and IFIP TC6.6 Workshop, EUNICE 2007 Proceedings*, Enschede, The Netherlands, pp.103 – 110.

[6] **Zou, J.**, **Xue, W.**, **Liang, Z. Zhao, Y.**, **Yang, B. and Shao, L.** (2007). *SIP* Parsing Offload: Design and Performance, *IEEE Global Telecommunications Conference. GLOBECOM'07*, Washington, DC, USA.

[7] **Hong, Y.**, **Huang, C. and Yan, J.** (2012). A Comparative Study of *SIP* Overload Control Algorithms, *Network and Traffic Engineering in Emerging Distributed Computing Applications*, 1 – 20, edited by J. Abawajy, M. Pathan, M. Rahman, A.K. Pathan, and M.M. Deris, IGI Global.

[8] **Hong, Y.**, **Huang, C. and Yan, J.** (2012). Impact of Retransmission Mechanism on *SIP* Overload: Stability Condition and Overload Control, *Journal of Networks*, *7*, 52 – 62.

[9] **Hong, Y.**, **Huang, C. and Yan, J.** (2012). Modeling and Design of a *Session Initiation Protocol* Overload Control Algorithm, *SIMULATION: Transactions of the Society for Modeling and Simulation International*, *88*(11), 1294 – 1315.

[10] **Handley, M.**, **Jacobson, V. and Perkins, C.** (2006). *SDP: Session Description Protocol*, RFC 4566 (Proposed Standard), July 2006. Obsoletes: 2327, 3266.

[11] **Schulzrinne, H.**, **Casner, S.**, **Frederick, R. and Jacobson, V.** (2003). *RTP*: A Transport Protocol for Real-Time Applications, RFC 3550 (Proposed Standard), July 2003. Obsoletes: 1889; updated by: 5506, 5761, 6051, 6222, 7022, 7160, 7164.

[12] **Baugher, M.**, **McGrew, D.**, **Naslund, M.**, **Carrara, E. and Norrman, K.** (2004). The *Secure Real-time Transport Protocol (SRTP)*, RFC 3711 (Proposed Standard), March 2004. Updated by: 5506, 6904.

[13] **Dierks, T. and Rescorla, E.** (2008). The *Transport Layer Security (TLS) Protocol* Version 1.2, RFC 5246 (Proposed Standard), August 2008. Obsoletes: 3268, 4346, 4366; updates: 4492; updated by: 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919.

[14] **Baset, S.**, **Gurbani, V.**, **Johnston, A.**, **Kaplan, H.**, **Rosen, B. and Rosenberg, J.** (2012). The *Session Initiation Protocol (SIP)*: An Evolutionary Study, *Journal of Communications*, *7*(2), 89–105.

[15] **Crocker, D. and Overell, P.** (2008). Augmented BNF for Syntax Specifications: ABNF, RFC 5234 (Internet Standard), January 2008. Obsoletes: 4234; updated by: 7405.

[16] **Sparks, R. and Zourzouvillys, T.** (2010). Correct Transaction Handling for 2xx Responses to *Session Initiation Protocol (SIP)* INVITE Requests, RFC 6026, September 2010.

[17] **Ohta, M.** (2006). Overload protection in a *SIP* signaling network, *In Proceedings of International Conference on Internet Surveillance and Protection.*, Cap Esterel, Cote d'Azur, France.

[18] **Noel, E. and Johnson, C.** (2007). Initial Simulation Results that Analyze *SIP* Based VoIP Networks Under Overload, *In Proceedings of 20th International Teletraffic Congress*, Ottawa, Canada.

[19] **Hilt, V. and Widjaja, I.** (2008). Controlling Overload in Networks of *SIP* Servers, *in Proc. of IEEE ICNP*, FL, USA, pp.83–93.

[20] **Noel, E. and Johnson, C.** (2009). Novel Overload Controls for *SIP* Networks, *In Proceedings of 21th International Teletraffic Congress*, Paris, France.

[21] **Mehta, A.**, **Hantehzadeh, N.**, **Gurbanit, V.**, **Hot, T.**, **Koshiko, J. and Viswanathan, R.** (2011). On the inefficacy of Euclidean classifiers for detecting self-similar *Session Initiation Protocol (SIP)* messages, *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, Dublin, Ireland.

[22] **Wei, X.**, **Zhiyong, L. and Bo, Y.** (2009). *SIP* Offload Engine for Accelerating J2EE Based *SIP* Application Server, *International Conference on Communication Software and Networks, ICCSN'09*, Macau, China.

[23] **Ohta, M.** (2009). Overload Control in a *SIP* Signalling Network, *International Journal of Electrical and Electronics Engineering*, 87 – 92.

[24] **Shen, C.**, **Schulzrinne, H. and Nahum, E.** (2008). *SIP* Server Overload Control: Design and Evaluation, *in Proceedings of IPTComm*, Heidelberg, Germany.

[25] **Garroppo, R.G.**, **Giordano, S.**, **Spagna, S. and Niccolini, S.** (2009). Queueing Strategies for Local Overload Control in *SIP* Server, *IEEE Global Telecommunications Conference*, Honolulu, Hawai, USA, pp.1 – 6.

[26] **Abaev, P.**, **Gaidamaka, Y.**, **Samouylov, K.**, **Pechinkin, A.**, **Razumchik, R. and Shorgin, R.** (2014). Hysteretic Control Technique for Overload Problem Solution in Network of *SIP* Servers, *Computing and Informatics*, *33*, 218 – 236.

[27] **Montagna, S. and Pignolo, M.** (2008). Performance evaluation of load control techniques in *SIP* signaling servers, *In Proceedings of ICONS 2008*, Cancun, Mexico, pp.51 – 56.

[28] **Azhari, S.**, **Homayouni, M.**, **Nemati, H.**, **Enayatizadeh, J. and Akbari, A.** (2012). Overload control in *SIP* networks using no explicit feedback: A window based approach, *Computer Communications*, *35*(12), 1472–1483.

[29] **Hilt, V.**, **Gurbani, V. and Schulzrinne, H.** (2014). *Session Initiation Protocol (SIP)* Overload Control, RFC 7339, September 2014.

[30] **Noel, E. and Williams, P.** (2015). *Session Initiation Protocol (SIP)* Rate Control, RFC 7415, February 2015.

[31] **ITU-T** (2004). ITU-T Recommendation I.371: Traffic control and congestion control in B-ISDN.

[32] **Shen, C.**, **Schulzrinne, H. and Koike, A.** (2014). A *Session Initiation Protocol (SIP)* Load-Control Event Package, RFC 7200, April 2014.

[33] **Samouylov, K.E.**, **Abaev, P.O.**, **Gaidamaka, Y.V.**, **Pechinkin, A.V. and Razumchik, R.V.** (2014). Analytical Modelling and Simulation for Performance Evaluation of *SIP* Server with Hysteretic Overload Control, *28th Conference on Modelling and Simulation*, Brescia, Italy.

[34] **Akbar, A.**, **Basha, S.M. and Sattar, S.A.** (2015). A Cooperative Overload Control Method for *SIP* Servers, *IEEE ICCSP 2015*, Melmaruvathur, India.

[35] **Jahanbakhsh, M.**, **Azhari, S.V.**, **Enayati-Zadeh, J. and Baghdadi, M.** (2016). Local and distributed *SIP* overload control solution improving sustainability of *SIP* networks, *Int. J. Commun. Syst.*, *29*(6), 1099–1131.

[36] **Mishra, G.**, **Dharmaraja, S. and Kar, S.** (2016). Reducing session establishment delay using timed out packets in *SIP* signaling network, *Int. J. Commun. Syst.*, *29*(2), 262–276.

[37] **Barakovic Husic, J.**, **Perenda, E.**, **Hadzialic, M. and Barakovic, S.** (2013). Performance Modelling and optimization of IP Multimedia Subsystem, *in Proc. UKSim–AMSS 7th European Modelling Symp. on Math. Modelling and Comput. Simulation (EMS2013)*, Manchester, United Kingdom.

[38] **Barakovic Husic, J.**, **Hidic, A.**, **Hadzialic, M. and Barakovic, S.** (2014). Simulation-based optimization of signaling procedures in IP multimedia subsystem, *In Proceedings of 15th FRUCT Conference*, St-Petersburg, Russia, pp.9–14.

[39] **Sun, J.**, **Yu, H. and Zheng, W.** (2008). Flow management with service differentiation for *SIP* application servers, *in: CHINAGRID '08: Proceedings of the Third China Grid Annual Conference, IEEE Computer Society*, Dunhuang, Gansu, China, pp.272–277.

[40] **Zhiguo, G.**, **Zhe, X.**, **Wei, X.**, **Zhiyong, L. and Bo, Y.** (2009). *SIP* Offload Engine for Accelerating J2EE Based *SIP* Application Server, *International Conference on Communication Software and Networks, ICCSN, 2009*, Macau, China.

[41] **Guduru, K.K. and Usha, J.** (2009). Queuing Strategies for Self Overload Control in *SIP* Servers, *Contemporary Computing and Informatics (IC3I), 2014 International Conference*, Mysore, India.

[42] **Camarillo, G.**, **Holmberg, C. and Gao, Y.** (2011). Re-INVITE and Target-Refresh Request Handling in the *Session Initiation Protocol (SIP)*, RFC 6141, March 2011.

[43] **Handley, M.**, **Schulzrinne, H.**, **Schooler, E. and Rosenberg, J.** (2010). *SIP: Session Initiation Protocol*, RFC 2543, March 1999.

[44] **Camarillo, G.**, **Schulzrinne, H. and Kantola, R.** (2003). Evaluation of Transport Protocols for the *Session Initiation Protocol*, *IEEE Network*, *17*(5), 40–46.

[45] **Walraevens, J.**, **Steyaert, B. and Bruneel, H.** (2003). Performance analysis of a single-server ATM queue with a priority scheduling, *Computers and Operations Research*, *30*(12), 1807–1829.

[46] **Maertens, T.**, **Walraevens, J. and Bruneel, H.** (2007). A modified HOL priority scheduling discipline: Performance analysis, *European Journal of Operational Research*, *180*(3), 1168–1185.

[47] **Maertens, T.**, **Walraevens, J. and Bruneel, H.** (2005). Analysis of a GI-1-1 queue with a modified HOL priority scheduling discipline, *Proc. of the COST 279 12th Management Committee Meeting*.

[48] **Sisalem, D.**, **Liisberg, M. and Rebahi, Y.** (2008). A Theoretical Model of the Effects of Losses and Delays on the Performance of *SIP*, *Globecom 2008*, New Orleans, USA.

[49] **Hong, Y.**, **Huang, C. and Yan, J.** (2011). Modeling and Simulation of *SIP* Tandem Server with Finite Buffer Algorithm, *ACM Trans. Model. Comput. Simul.*, *21*(2), article 11.

[50] **Schwartz, M.** (1996). *Broadband Integrated Networks*, Prentice-Hall, Inc.

[51] **Revzina, J.** (2010). Possibilities of MMPP Processes for Bursty Traffic Analysis, *Proceedings of the 10th International Conference "Reliability and Statistics in Transportation and Communication"*, Riga, Latvia, pp.131–135.

[52] **Abaev, P.**, **Khachko, A. and Beschastny, V.** (2014). Queuing Model for *SIP* Server Hysteretic Overload Control with K-state MMPP Bursty Traffic, *Applied Problems in Theory of Probabilities and Mathematical Statistics Related to Modeling of Information Systems*, St-Petersburg, Russia, pp.495–500.

[53] **Muscarielloa, L.**, **Melliaa, M.**, **Meoa, M.**, **Ajmone Marsana, M. and Lo Cigno, R.** (2005). Markov models of internet traffic and a new hierarchical MMPP model, *Computer Communications*, *28*, 1835–1851.

[54] **Bali, S.**, (2004), Fitting MMPP to IP Stream, `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.3862&rep=rep1&type=pdf`, retrieved date: 01-December-2016.

[55] **Hong, Y.**, **Huang, C. and Yan, J.** (2010). Analysis of *SIP* Retransmission Probability Using a Markov-Modulated Poisson Process Model, *in Proc. of IEEE Network Operations and Management Symposium (NOMS)*, Osaka, Japan, pp.179–186.

[56] **Zhou, Y. and Gans, N.**, (2005), A Single-Server Queue with Markov Modulated Service Times, Working paper, The Wharton School, University of Pensilvania. [Online Available]. Available: `http:fic.wharton.upenn.edu/fic`, retrieved date: 07 December 2016.

[57] **Zukerman, M.**, (2016), Introduction to Queueing Theory and Stochastic Teletraffic Models, `http://arxiv.org/pdf/1307.2968.pdf`, lecture notes, 9 April 2016 [Accessed: 25-October-2016].

[58] **Baykal-Gursoy, M. and Xiao, W.** (2004). Stochastic Decomposition in M/M/∞ Queues with Markov Modulated Service Rates, *Queueing Systems*, *48*(1), 75–88.

[59] **Revzina, J.** (2010). Performance of MMPP Processes for Bursty Traffic Analysis, *Proceedings of the 10th International Conference "Reliability and Statistics in Transportation and Communication" (RelStat'10)*, Riga, Latvia, pp.131–135.

[60] **Yavas, D.**, `http://web.itu.edu.tr/~demiryavas/software/MMPP/MMPP_M_1_N_Resolver.java`, retrieved date: 30.11.2016.

[61] **Yavas, D.**, `http://web.itu.edu.tr/~demiryavas/software/MMPP/M_MMPP_1_N_Resolver.java`, retrieved date: 30.11.2016.

[62] **Adan, I. and Resing, J.**, (2010), Discrete-event simulation of single-stage production system, Sofware. [Online Available]. `http://www.win.tue.nl/~iadan/blockq/mm1.zip`, retrieved date: 07 December 2016.

[63] **Cargal, J.**, (1988), Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff, [Chapter 32: Recursion and Averages, Online Available]. http://www.cargalmathbooks.com/32%20Averages%20.pdf, retrieved date: 07 January 2017.

[64] **Ruhm, K.**, Recursive Averaging, [Online Available]. http://www.mmm.ethz.ch/dok01/d0000560.pdf, retrieved date: 07 January 2017.

[65] **Yavas, D.**, http://web.itu.edu.tr/~demiryavas/, retrieved date: 30.11.2016.

[66] **Knuth, D.** (1997). *The Art of Computer Programming, Volume 2 Seminumerical Algorithms*, Addison-Wesley.

[67] **Url-1 JAIN-SIP**, https://jsip.java.net/, retrieved date: 23-October-2016.

[68] **Url-2 SIPp Traffic Generator**, http://sipp.sourceforge.net/, retrieved date: 5-December-2016.

[69] **Perea, R.** (2008). *Internet Multimedia Communications Using* SIP*: A Modern Approach Including Java Practice*, Elsevier, Inc.

APPENDICES

APPENDIX A.1 : Derivation of Conventional SIP Server Model Formulation for Infinite Buffer Case
APPENDIX B.1 : M/MMPP(2)/1/N Queuing Model

APPENDIX A.1: Derivation of Conventional SIP Server Model for Infinite Buffer Case

Hong, Y. et al. [49] [9] derived the discrete-time fluid-flow based analytical model for the conventional SIP server with finite buffer. We modify the model for a single proxy SIP server by simplifying the considerations such as ignoring the queuing of response messages from the downstream server. Hence, we consider a discrete-time queuing model with single server equipped with a finite buffer, as shown in the Figure A.1. Assumptions for the model are similar to the infinite buffer case.

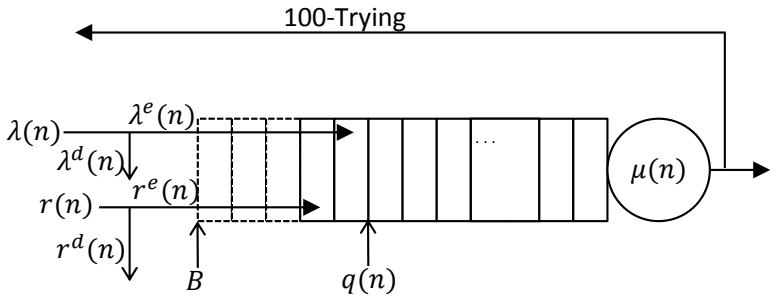


Figure A.1 : Queuing model for the conventional SIP with finite buffer.

Recalling basic definitions from Section 4.3, dynamics of the queuing model of the conventional SIP server are characterized by solely two processes; the arrival and service processes. The arrival process is determined by an aggregation of original and retransmitted request message rates, denoted by $\lambda(n)$ and $r(n)$ for time slot n, respectively. The service process determined by the service time of an INVITE message denoted by $\mu(n)$.

Since the server has finite buffer size B, the incoming arrivals including original and retransmitted messages are blocked and lost when the buffer is full i.e. $(q(n) = B)$. In the current time slot n, the original invite requests arrive with a rate $\lambda(n)$, only $\lambda^e(n)$, which are called effective original arrival messages, enter the buffer of the server and $\lambda^d(n)$ are dropped. Similarly the retransmitted invite requests arrive with a rate $r(n)$, only $r^e(n)$, which are called effective retransmitted arrival messages, enter the buffer of the server and $r^d(n)$ are dropped because of the blocking. The server can process $\mu(n)$ request messages. Both $\lambda(n)$ and $\mu(n)$ can be arbitrary stochastic processes. $\lambda^e(n)$, $\lambda^d(n)$, and $r^d(n)$ can be calculated with Eqs. (4.18), (4.19) and (4.21), respectively. Effective retransmissions, $r^e(n)$, can be calculated as follows:

$$r^e(n) = min\{B - q(n) + \mu(n) - \lambda^e(n), r(n)\}. \tag{A.1}$$

Then, the queue size of the server at next time slot $n+1$ can be obtained as based on the information at the current time slot n:

$$q(n+1) = \left[min\{q(n) + \lambda^e(n) + r^e(n) - \mu(n), B\}\right]^+ \tag{A.2}$$

1.1 Modeling retransmissions

By considering triangle representation of decomposed retransmissions, therefore Eq.(4.1) and Eq.(4.25) are still valid for the conventional SIP case, decomposed retransmissions can be calculated through the following formulas recursively:

$$r_{j1}(n) = min\left\{ \left[\lambda^e(n-T_j) + q(n-T_j) - z(n-T_j,n)\right]^+, \lambda^e(n-T_j) \right\} \tag{A.3}$$

where $z(m,n) = \sum_{l=m}^{n} \mu(l)$.

$$r^e_{j1}(n) = min\left\{ \left[B + \mu(n) - q(n) - \lambda^e(n) - \sum_{l=1}^{j-1} r_l(n)\right]^+, r_{j1}(n) \right\} \tag{A.4}$$

Eq.(4.28) can be used to calculate $r_{jj+1}(n)$ for the conventional SIP case.

$$\begin{aligned} r_{ji}(n) = min\Big\{ & \left[r^e_{i-1i}(n-T_{ji-1}) + q(n-T_{ji-1}) + R_i(n-T_{ji-1}) \right. \\ & \left. - z(n-T_{ji-1},n)\right]^+, \quad r^e_{i-1i}(n-T_{ji-1}) \Big\} \end{aligned} \tag{A.5}$$

where; $R_i(m) = \sum_{l=1}^{i-2} r^e_l(m) + \sum_{l=1}^{i-1} r^e_{i-1l}(m)$, which denote the aggregated retransmissions joined the queue prior to $r^e_{i-1i}(m)$ and $r^e_0(m) = 0$.

$$\begin{aligned} r^e_{i-1i}(n-T_{ji-1}) = min\Big\{ & \left[B - q(n-T_{ji-1}) - \lambda^e(n-T_{ji-1}) + \mu(n-T_{ji-1}) \right. \\ & \left. - R_i(n-T_{ji-1})\right]^+, r_{i-1i}(n-T_{ji-1}) \Big\} \end{aligned} \tag{A.6}$$

$$r^e_{ji}(n) = min\left\{ \left[B - q(n) - \lambda^e(n) + \mu(n) - R_i(n)\right]^+, r_{ji}(n) \right\} \tag{A.7}$$

APPENDIX B.1: M/MMPP(2)/1/N Queuing Model

This model assumes two modes of operations in the SIP server, normal service mode with a mean service rate μ_1 and maintenance mode with mean service rate μ_0. This model is assumed at [55] and normal and maintenance operation modes are assumed. We will use same terminology as in this paper [55]. Maintenance mode is assumed as server's routine maintenance mode such as required periodic database synchronization.

The mean time for normal service is $1/\delta_1$ while the mean time for the maintenance is $1/\delta_0$, and both assumed to be exponential distributed. Then the system leaves the normal service mode with a mean rate δ_1 while it leaves the maintenance mode with a mean rate δ_0. The SIP traffic arrives in the server with a mean rate λ and the arrival rate is assumed to be Poisson distributed. Transition between the normal service mode and the maintenance mode can be governed by an underlying continuous Markov chain.

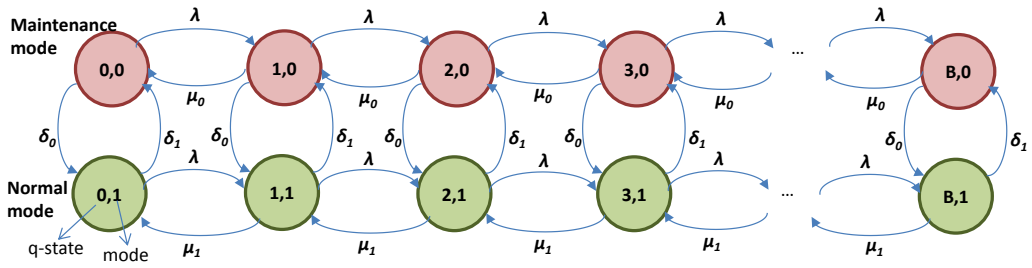


Figure B.1 : M/MMPP/1/B Queue State Transition Diagram.

The paper [55] assumes the SIP server with infinite queue. We assume the server as finite queue sized and developed analytical solutions by using the approach suggested at [57]. The state transition diagram of the considered M/MMPP(1)/1/B queue is shown in Figure B.1. While the analytical solution suggested at [57] is for MMPP(2)/M/1/B queues, we determine rules for infinitesimal matrix, $\boldsymbol{Q}$, for M/MMPP(2)/1/B queues as follows:

$$Q_{00,00} = -\lambda - \delta_0; \quad Q_{00,01} = \delta_0; \quad Q_{00,10} = \lambda;$$

$$Q_{01,00} = \delta_1; \quad Q_{01,01} = -\lambda - \delta_1; \quad Q_{01,11} = \lambda$$

for $B > i > 0$, the nonzero entries

$$Q_{i0,i0} = -\lambda - \delta_0 - \mu_0; \quad Q_{i0,i1} = \delta_0; \quad Q_{i0,(1+1)0} = \lambda$$

$$Q_{i1,i0} = \delta_1; \quad Q_{i1,i1} = -\lambda - \delta_1 - \mu_1; \quad Q_{11,(i+1)1} = \lambda$$

and

$$Q_{B0,(B-1)0} = \mu_0; \quad Q_{B0,B0} = -\delta_0 - \mu_0; \quad Q_{B0,B1} = \delta_0$$

$$Q_{B1,(B-1)1} = \mu_1; \quad Q_{B1,B1} = -\delta_1 - \mu_1; \quad Q_{B1,B0} = \delta_1$$

There is also the following normalizing equation given:

$$\sum_{i=0}^{B}\sum_{m=0}^{1}\pi_{im}=1 \tag{B.1}$$

By using rules above, the equation $0=\boldsymbol{\Pi Q}$ can be written in detailed form.

$$0=\begin{bmatrix}\pi_{00} & \pi_{01} & \pi_{10} & \dots\end{bmatrix}\times$$
$$\begin{bmatrix}
-\lambda-\delta_0 & \delta_0 & \lambda & 0 & 0 & 0 & \dots\\
\delta_1 & -\lambda-\delta_1 & 0 & \lambda & 0 & 0 & \dots\\
\mu_0 & 0 & -\lambda-\delta_0-\mu_0 & \delta_0 & \lambda & 0 & \dots\\
0 & \mu_1 & \delta_1 & -\lambda-\delta_1-\mu_1 & 0 & \lambda & \dots\\
0 & 0 & \mu_0 & 0 & -\lambda-\delta_0-\mu_0 & \delta_0 & \dots\\
0 & 0 & 0 & \mu_1 & \delta_1 & -\lambda-\delta_1-\mu_1 & \dots\\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}$$

Steady-state equations can be calculated by using Gauss-Seidel, successive approximation method [57]. A software program, written in Java, using this method is given in [61]. Once steady-state probabilities are calculated, queuing dynamics can be calculated accordingly. The blocking probability can be calculated as follows:

$$P_b=\pi_B=\pi_{B0}+\pi_{B1} \tag{B.2}$$

The steady-state average service rate can be calculated as follows [56, 57]:

$$\frac{1}{\mu_{av}}=\frac{\delta_1}{\delta_0+\delta_1}\frac{1}{\mu_0}+\frac{\delta_0}{\delta_0+\delta_1}\frac{1}{\mu_1} \tag{B.3}$$

CURRICULUM VITAE

Name Surname: Demir Yaşa Yavaş

Place and Date of Birth: İzmir, 04/12/1961

Adress: Çınar Mah. Başefendi Sokak, No:33/9 Küçükyalı İstanbul

E-Mail: demiryavas@itu.edu.tr, demiry@netas.com.tr, demir.yavas@yahoo.com

B.Sc.: Istanbul Technical University, Faculty of Electrical and Electronic Engineering, Department of Electronics and Communication Engineering

M.Sc.: Istanbul Technical University, Graduate School of Science, Engineering and Technology, Telecommunication Engineering Graduate Program

Professional Experience and Rewards:

- Research Assistant in Istanbul Technical University, Faculty of Electrical and Electronic Engineering, Department of Telecommunication (1985-1989)
- Employed in Netas (a joint venture company with NORTEL&GENBAND which are the companies producing Telecommunication Switches and IP multimedia products) as Software Designer, Design Department Manager, System Engineer and Software Architect, respectively. (1990-current)

List of Publications and Patents:

PUBLICATIONS/PRESENTATIONS ON THE THESIS

Journal Papers

- **Yavas D.Y.**, Hokelek I. and Gunsel B., 2014: Strict Prioritization of New Requests over Retransmissions for Enhancing Scalability of SIP Servers. *IEICE Transactions on Communications*, Vol. E97-B, No.12, Dec. 2014.
- **Yavas D.Y.**, Hokelek I. and Gunsel B., 2016: On Modeling of Priority-Based SIP Request Scheduling. *Simulation Modelling Practice and Theory (SIMPAT)*, (submitted: 16 August 2016 (under review)).
- **Yavas D.Y.**, Hokelek I. and Gunsel B., 2016: On Fluid-Flow Modeling of Priority Based Request Scheduling for Finite Buffer SIP Server. *International Journal of Communication Systems*, (submitted: 14 November 2016 (under review)).

International Conference Papers

- **Yavas D.Y.**, Hokelek I. and Gunsel B., 2013: An Analysis Tool to Evaluate Effect of Retransmissions on SIP Server Overloading. *In Proc. of 7th Int. Conference*

on Next Generation Mobile Apps, Services and Technologies (NGMAST), pp.99-104, September 25-27, 2013, Prague, Czech Republic

- **Yavas D.Y.**, Hokelek I. and Gunsel B., 2015: Controlling SIP Server Overload with Priority Based Request Scheduling. *International Conference on Computing, Networking and Communications (ICNC)*, February 16-19, 2015, Anaheim, California, USA.

- **Yavas D.Y.**, Hokelek I. and Gunsel B., 2016: Analytical Model of Priority Based Request Scheduling Mechanism Preventing SIP Server Overload. *Premier International Conference for Military Communications (MILCOM)*, November 1-3, 2016, Baltimore, MD, USA.

- **Yavas D.Y.**, Hokelek I. and Gunsel B., 2016: Modeling of Priority-based Request Scheduling Mechanism for Finite Buffer SIP Servers. *11th International Conference on Queueing Theory and Network Applications (QTNA2016)*, December 13-15, 2016, Wellington, New Zealand.

National Conference Papers

- **Yavas D.Y.**, Hokelek I. and Gunsel B., 2015: SIP Sunucu Yüklenme Analiz Aracı (Analysis Tool for SIP Server Loading). *IEEE 23. Sinyal İşleme ve İletişim Uygulamaları Kurultayı (SIU)*, May 16-19, 2015, Malatya, Turkey.

PATENTS SUBMISSIONS ON THE THESIS

National Patent Applications

- SIP Sunucularının Aşırı Yüklenmesini Önlemek Üzere Bir Yöntem ve Sistem, PT2015-01353 Başvuran Kurum: NETAŞ, 2015.

- SIP Protokolünde Uygulama Seviyesi Tekrar İletim İçin Bir Yöntem, PT2015-01271 Başvuran Kurum: NETAŞ, 2015.