

THE PERFORMANCE OF CLUSTERING TECHNIQUES FOR SCALABLE WEB SERVERS

by

Liang Zhang

B.Sc., Zhejiang University, 1995

M.Sc., Simon Fraser University 1999

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Master of Applied Science

in the School of
Engineering Science

© Liang Zhang 2002

SIMON FRASER UNIVERSITY

August 2002

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without permission of the author.

Approval

Name: Liang Zhang
Degree: Master of Applied Science
Title of thesis: The Performance of Clustering Techniques for Scalable Web
Servers

Examining Committee:

Chair: Dr. Paul Ho

Dr. Jacques Vaisey, P.Eng.
Associate Professor
School of Engineering Science

Dr. Shawn Stapleton
Professor
School of Engineering Science

Dr. Tejinder Randhawa
External Examiner

Date Approved: _August 1st., 2002_

Abstract

Clustering is a technique for extending the capacity of web-servers by sharing the work over multiple machines that share the same external address. Many different strategies are possible for distributing the requests among the cluster members; however, finding a strategy that works well over a wide range of load distributions and network configurations is difficult. Recent research has used real traffic traces, wide-area testbeds or small-scale labs to compare the performance of different architectures and load distribution algorithms. In practice, however, these methods are time and resource consuming. In this thesis, we present an efficient and inexpensive framework to study Web clustering technologies via simulation models that are based upon stress tests on real servers. The method is then applied to compare the performance of two server architectures and five load-balancing algorithms. Our results show that clustered servers can achieve performance levels comparable to a high-end computer but at lower costs and maximum resource efficiency. For the types of traffic considered, simple load balancing algorithms in a “switch” architecture, in which replies are sent directly to clients rather than through the load balancer, achieve the best performance in both response time and load balancing. The top performing scheduling algorithm is the Least-Connection scheme in which the load balancing decision is based on the number of active connections.

Dedication

To my parents and brother.

Acknowledgments

The author is deeply grateful to my senior supervisor, Dr. Jacques Vaisey, for all the advice in the past three years, especially for his invaluable guidance and for the time he spent working with me throughout the development of my thesis.

Thanks also go to Mr. Eric Delisle for his effort in proof-reading my thesis.

Finally, the author acknowledges with gratitude, the constant encouragement and support from my mother, father, and elder brother.

Table of Contents

Approval	ii
Abstract.....	iii
Dedication.....	iv
Acknowledgments	v
Table of Contents	vi
List of Tables	ix
List of Figures	xv
List of Abbreviations	xviii
1 Introduction	1
1.1 Thesis Goals	
1.2 Summary of Contributions	
1.3 Thesis Outline	
2 Techniques for Scalable Web Server Clustering	6
2.1 OSI Reference Model and TCP/IP Network Model	
2.1.1 The TCP/IP Protocol	
2.1.2 Overview of HTTP Protocol	
2.2 Cluster-Based Scalable Service Architectures	
2.2.1 Server Selection Strategies	
2.2.2 Advantages and Disadvantages	
2.3 Scheduling Algorithms	
2.4 Distributed Packet Rewriting	
2.5 Summary	

3	Traffic Model and Performance Evaluation of Web Servers	37
3.1	Parameters Used to Evaluate Web Server Performance	
3.2	Web Traffic Characteristics	
3.2.1	Distributional characterization of the Web Traffic	
3.3	Workload Generator of Web Traffic	
3.4	Web Benchmarks	
3.5	Summary	
4	Server Stress Tests and Server Models	55
4.1	Experimental Setup	
4.2	Assumptions and Restrictions	
4.3	Server Stress-test Results	
4.4	Development of the Server Models	
4.4.1	Server Models	
4.5	Integration and Validation of the Server Models	
4.5.1	Integration of the Server Models	
4.5.2	Server Validation	
4.6	Summary	
5	Simulations and Results	82
5.1	Simulation Configurations	
5.1.1	Server Model	
5.1.2	Load Balancer Model	
5.1.3	Network Model	
5.2	Simulation Results	
5.2.1	Alternative Implementations	

5.2.2 Latencies and Throughputs	
5.2.3 Load Balancing	
5.2.4 The Effect of Average File Sizes	
5.2.5 The Effect of Number of Servers in a Cluster	
5.2.6 The Effect of Network Performance Parameters	
5.2.7 The Impact of Averaging Windows	
5.3 Summary	
6 Conclusions	117
Appendices	123
A .Simulation	
B. SURGE Measurements	
C. Server Models (Integration and Validation)	
D. Simulation Results	
E. Stem-and-Leaf Plot for Response Times	
Bibliography	202

List of Tables

3.1 Probability density functions (pdfs) of three models	41
3.2 Distributions and parameters of several Web characteristics used in this project	45
4.1 Experiment specifications	56
4.2 Apache 1.3.12 configuration parameters	58
4.3 File sizes generated by SURGE	60
4.4 Measured and predicted mean latencies of Web objects	68
4.5 Parameters for polynomial fittings of the server models $1 \leq n \leq 140$	69
4.6 Parameters for polynomial fittings of the server models $140 < n \leq 960$	69
4.7 Parameters for polynomial fittings of the percentage of denied requests	70
5.1 Summary statistics ($\mathbf{s}=1.143, N_{UE} = 640$)	92
5.2 Summary statistics for connections ($\mathbf{s}=1.143, N_{UE} = 640$)	97
5.3 Summary statistics for connections ($\mathbf{s}=2.5, N_{UE} = 640$)	97
5.4 Summary statistics ($\mathbf{s}=1.143, N_{UE} = 640$)	109
5.5 Summary statistics for connections XSIZE-G, $\mathbf{s}=1.143, N_{UE} = 640$	109
5.6 Summary statistics ($\mathbf{s}=1.143, N_{UE} = 640$)	114
5.7 Summary statistics for connections RT-G, $\mathbf{s}=1.143, N_{UE} = 640$	109
B.1 Roundtrip time measured by “traceroute” command	125
B.2 Bandwidth measured by FTP requests	125
B.3 Measured latencies of Web objects from two Apache configurations	126
B.4 Average latencies for $\mathbf{s}=1.143$	126
B.5 Byte throughputs for $\mathbf{s}=1.143$	127
B.6 HTTP ops/second for $\mathbf{s}=1.143$	127
B.7 Object throughputs for $\mathbf{s}=1.143$	128
B.8 Numbers of timeouts for $\mathbf{s}=1.143$	128
B.9 Average latencies for $\mathbf{s}=1.7145$	129
B.10 Byte throughputs for $\mathbf{s}=1.7145$	129
B.11 HTTP ops/second for $\mathbf{s}=1.7145$	130
B.12 Object throughputs for $\mathbf{s}=1.7145$	130
B.13 Numbers of timeouts for $\mathbf{s}=1.7145$	131

B.14 Average latencies for $s=2.5$	131
B.15 Byte throughputs for $s=2.5$	132
B.16 HTTP ops/second for $s=2.5$	132
B.17 Object throughputs for $s=2.5$	133
B.18 Numbers of timeouts for $s=2.5$	133
B.19 Throughput and the number of requests rejected ($s=1.143$)	134
B.20 Throughput and the number of requests rejected ($s=1.7145$)	134
B.21 Throughput and the number of requests rejected ($s=2.5$)	135
C.1 Average latencies, $s=1.143$, 10 Mbps	136
C.2 Object throughputs, $s=1.143$, 10 Mbps	137
C.3 Byte throughputs, $s=1.143$, 10 Mbps	137
C.4 Average latencies, $s=1.7145$, 10 Mbps	138
C.5 Object throughputs, $s=1.7145$, 10 Mbps	138
C.6 Byte throughputs, $s=1.7145$, 10 Mbps	139
C.7 Average latencies, $s=2.5$, 10 Mbps	139
C.8 Object throughputs, $s=2.5$, 10 Mbps	140
C.9 Byte throughputs, $s=2.5$, 10 Mbps	140
C.10 Average latencies, $s=1.143$, 100 Mbps	141
C.11 Object throughputs, $s=1.143$, 100 Mbps	141
C.12 Byte throughputs, $s=1.143$, 100 Mbps	142
C.13 Average latencies, $s=1.7145$, 100 Mbps	142
C.14 Object throughputs, $s=1.7145$, 100 Mbps	143
C.15 Byte throughputs, $s=1.7145$, 100 Mbps	143
C.16 Average latencies, $s=2.5$, 100 Mbps	144
C.17 Object throughputs, $s=2.5$, 100 Mbps	144
C.18 Byte throughputs, $s=2.5$, 100 Mbps	145
D.1 Average latencies for RR-G (4-server), $s=1.143$	146
D.2 Object throughputs for RR-G (4-server), $s=1.143$	147
D.3 Byte throughputs for RR-G (4-server), $s=1.143$	147
D.4 Average latencies for XSIZE-G (4-server), $s=1.143$, averaging window = 1s	148
D.5 Object throughputs for XSIZE-G (4-server), $s=1.143$, averaging window = 1s	148

D.6 Byte throughputs for XSIZE-G (4-server), $s=1.143$, averaging window = 1s	149
D.7 Average latencies for CONN-G (4-server), $s=1.143$, averaging window = 1s	149
D.8 Object throughputs for CONN-G (4-server), $s=1.143$, averaging window = 1s	150
D.9 Byte throughputs for CONN-G (4-server), $s=1.143$, averaging window = 1s	150
D.10 Average latencies for RT-G (4-server), $s=1.143$, averaging window = 1s	151
D.11 Object throughputs for RT-G (4-server), $s=1.143$, averaging window = 1s	151
D.12 Byte throughputs for RT-G (4-server), $s=1.143$, averaging window = 1s	152
D.13 Average latencies for RAN-G (4-server), $s=1.143$	152
D.14 Object throughputs for RAN-G (4-server), $s=1.143$	153
D.15 Byte throughputs for RAN-G (4-server), $s=1.143$	153
D.16 Average latencies for RR-S (4-server), $s=1.143$	154
D.17 Object throughputs for RR-S (4-server), $s=1.143$	154
D.18 Byte throughputs for RR-S (4-server), $s=1.143$	155
D.19 Average latencies for XSIZE-S (4-server), $s=1.143$, averaging window = 1s	155
D.20 Object throughputs for XSIZE-S (4-server), $s=1.143$, averaging window = 1s	156
D.21 Byte throughputs for XSIZE-S (4-server), $s=1.143$, averaging window = 1s	156
D.22 Average latencies for CONN-S (4-server), $s=1.143$, averaging window = 1s	157
D.23 Object throughputs for CONN-S (4-server), $s=1.143$, averaging window = 1s	157
D.24 Byte throughputs for CONN-S (4-server), $s=1.143$, averaging window = 1s	158
D.25 Average latencies for RT-S (4-server), $s=1.143$, averaging window = 1s	158
D.26 Object throughputs for RT-S (4-server), $s=1.143$, averaging window = 1s	159
D.27 Byte throughputs for RT-S (4-server), $s=1.143$, averaging window = 1s	159
D.28 Average latencies for RAN-S (4-server), $s=1.143$	160
D.29 Object throughputs for RAN-S (4-server), $s=1.143$	160
D.30 Byte throughputs for RAN-S (4-server), $s=1.143$	161
D.31 Average latencies for RR-G (4-server), $s=2.5$	161
D.32 Object throughputs for RR-G (4-server), $s=2.5$	162
D.33 Byte throughputs for RR-G (4-server), $s=2.5$	162
D.34 Average latencies for XSIZE-G (4-server), $s=2.5$, averaging window = 1s	163
D.35 Object throughputs for XSIZE-G (4-server), $s=2.5$, averaging window = 1s	163
D.36 Byte throughputs for XSIZE-G (4-server), $s=2.5$, averaging window = 1s	164

D.37 Average latencies for CONN-G (4-server), $s=2.5$, averaging window = 1s	164
D.38 Object throughputs for CONN-G (4-server), $s=2.5$, averaging window = 1s	165
D.39 Byte throughputs for CONN-G (4-server), $s=2.5$, averaging window = 1s	165
D.40 Average latencies for RT-G (4-server), $s=2.5$, averaging window = 1s	166
D.41 Object throughputs for RT-G (4-server), $s=2.5$, averaging window = 1s	166
D.42 Byte throughputs for RT-G (4-server), $s=2.5$, averaging window = 1s	167
D.43 Average latencies for RAN-G (4-server), $s=2.5$	167
D.44 Object throughputs for RAN-G (4-server), $s=2.5$	168
D.45 Byte throughputs for RAN-G (4-server), $s=2.5$	168
D.46 Average latencies for RR-S (4-server), $s=2.5$	169
D.47 Object throughputs for RR-S (4-server), $s=2.5$	169
D.48 Byte throughputs for RR-S (4-server), $s=2.5$	170
D.49 Average latencies for XSIZE-S (4-server), $s=2.5$, averaging window = 1s	170
D.50 Object throughputs for XSIZE-S (4-server), $s=2.5$, averaging window = 1s	171
D.51 Byte throughputs for XSIZE-S (4-server), $s=2.5$, averaging window = 1s	171
D.52 Average latencies for CONN-S (4-server), $s=2.5$, averaging window = 1s	172
D.53 Object throughputs for CONN-S (4-server), $s=2.5$, averaging window = 1s	172
D.54 Byte throughputs for CONN-S (4-server), $s=2.5$, averaging window = 1s	173
D.55 Average latencies for RT-S (4-server), $s=2.5$, averaging window = 1s	173
D.56 Object throughputs for RT-S (4-server), $s=2.5$, averaging window = 1s	174
D.57 Byte throughputs for RT-S (4-server), $s=2.5$, averaging window = 1s	174
D.58 Average latencies for RAN-S (4-server), $s=2.5$	175
D.59 Object throughputs for RAN-S (4-server), $s=2.5$	175
D.60 Byte throughputs for RAN-S (4-server), $s=2.5$	176
D.61 Average latencies for RR-G (2-server), $s=1.143$	176
D.62 Object throughputs for RR-G (2-server), $s=1.143$	177
D.63 Byte throughputs for RR-G (2-server), $s=1.143$	177
D.64 Average latencies for RR-G (8-server), $s=1.143$	178
D.65 Object throughputs for RR-G (8-server), $s=1.143$	178
D.66 Byte throughputs for RR-G (8-server), $s=1.143$	179
D.67 Average latencies for XSIZE-G (4-server), $s=1.143$, averaging window = 0.1s	179

D.68 Object throughputs for XSIZE-G (4-server), $s=1.143$, averaging window = 0.1s	180
D.69 Byte throughputs for XSIZE-G (4-server), $s=1.143$, averaging window = 0.1s	180
D.70 Average latencies for XSIZE-G (4-server), $s=1.143$, averaging window = 4s	181
D.71 Object throughputs for XSIZE-G (4-server), $s=1.143$, averaging window = 4s	181
D.72 Byte throughputs for XSIZE-G (4-server), $s=1.143$, averaging window = 4s	182
D.73 Average latencies for XSIZE-G (4-server), $s=1.143$, averaging window = 8s	182
D.74 Object throughputs for XSIZE-G (4-server), $s=1.143$, averaging window = 8s	183
D.75 Byte throughputs for XSIZE-G (4-server), $s=1.143$, averaging window = 8s	183
D.76 Average latencies for CONN-G (4-server), $s=1.143$, averaging window = 0.1s	184
D.77 Object throughputs for CONN-G (4-server), $s=1.143$, averaging window = 0.1s	184
D.78 Byte throughputs for CONN-G (4-server), $s=1.143$, averaging window = 0.1s	185
D.79 Average latencies for CONN-G (4-server), $s=1.143$, averaging window = 4s	185
D.80 Object throughputs for CONN-G (4-server), $s=1.143$, averaging window = 4s	186
D.81 Byte throughputs for CONN-G (4-server), $s=1.143$, averaging window = 4s	186
D.82 Average latencies for CONN-G (4-server), $s=1.143$, averaging window = 8s	187
D.83 Object throughputs for CONN-G (4-server), $s=1.143$, averaging window = 8s	187
D.84 Byte throughputs for CONN-G (4-server), $s=1.143$, averaging window = 8s	188
D.85 Average latencies for RT-S (4-server), $s=1.143$, averaging window = 0.1s	188
D.86 Object throughputs for RT-S (4-server), $s=1.143$, averaging window = 0.1s	189
D.87 Byte throughputs for RT-S (4-server), $s=1.143$, averaging window = 0.1s	189
D.88 Average latencies for RT-S (4-server), $s=1.143$, averaging window = 4s	190
D.89 Object throughputs for RT-S (4-server), $s=1.143$, averaging window = 4s	190
D.90 Byte throughputs for RT-S (4-server), $s=1.143$, averaging window = 4s	191
D.91 Average latencies for RT-S (4-server), $s=1.143$, averaging window = 8s	191
D.92 Object throughputs for RT-S (4-server), $s=1.143$, averaging window = 8s	192
D.93 Byte throughputs for RT-S (4-server), $s=1.143$, averaging window = 8s	192
D.94 Average latencies for Bryhni-CONN-G-1.1 (4-server), $s=1.143$	193
D.95 Object throughputs for Bryhni-CONN-G-1.1 (4-server), $s=1.143$	193
D.96 Byte throughputs for Bryhni-CONN-G-1.1 (4-server), $s=1.143$	194
D.97 Average latencies for Bryhni-CONN-G-1.0 (4-server), $s=1.143$	194
D.98 Object throughputs for Bryhni-CONN-G-1.0 (4-server), $s=1.143$	195

D.99 Byte throughputs for Bryhni-CONN-G-1.0 (4-server), $\mathbf{s} = 1.143$	195
D.100 Average latencies for Bryhni-RT-G-1.0 (4-server), $\mathbf{s} = 1.143$	196
D.101 Object throughputs for Bryhni-RT-G-1.0 (4-server), $\mathbf{s} = 1.143$	196
D.102 Byte throughputs for Bryhni-RT-G-1.0 (4-server), $\mathbf{s} = 1.143$	197
D.103 Average latencies for Bryhni-XSIZE-S-1.0 (4-server), $\mathbf{s} = 1.143$	197
D.104 Object throughputs for Bryhni-XSIZE-S-1.0 (4-server), $\mathbf{s} = 1.143$	198
D.105 Byte throughputs for Bryhni-XSIZE-S-1.0 (4-server), $\mathbf{s} = 1.143$	198
E.1 Stem-and-leaf plot for the response times (XSIZE-G)	199
E.2 Stem-and-leaf plot for the response times (RR-G)	200
E.3 Stem-and-leaf plot for the response times (CONN-G)	200
E.4 Stem-and-leaf plot for the response times (RT-G)	201
E.5 Stem-and-leaf plot for the response times (RAN-G)	201

List of Figures

1.1 Illustration of clustered servers	3
2.1 Network architecture based on the OSI reference model	7
2.2 Network architectures based on the OSI reference model and TCP/IP protocol	10
2.3 Packet exchanges and round-trip times for HTTP/1.1	13
2.4 The server-side URL redirection	16
2.5 Clustering via NAT	17
2.6 Illustration of clustering via NAT	19
2.7 Clustering via IP tunneling or IP modification	21
2.8 Round robin DNS	26
2.9 Hybrid scheduling	30
2.10 Content-based scheduling	32
2.11 Master-slave scheme	34
2.12 Distributed packet rewriting	35
3.1 Typical workload generator	48
4.1 Experiment setup	57
4.2 Comparison of latencies for two Apache configurations ($s=1.143$)	61
4.3 Comparison of latencies for two Apache configurations ($s=1.7145$)	61
4.4 Comparison of latencies for two Apache configurations ($s=2.5$)	62
4.5 Latencies for Apache 1.2.13 (Experiment II)	62
4.6 Data throughputs (bytes/second)	64
4.7 Object throughputs (objects/second)	64
4.8 Percentage of denied requests	65
4.9 Regression fittings for $s=1.143$	66
4.10 Regression fittings for $s=1.7145$	67
4.11 Regression fittings for $s=2.5$	67
4.12 Regression fittings for 3 s 's used	68
4.13 Simplified network architecture for server processing models	72
4.14 Flow chart of the server model implementation	76
4.15 Comparisons of measured and simulated average latencies	77

4.16 Comparisons of measured and simulated object throughputs	78
4.17 Comparisons of measured and simulated byte throughputs	78
4.18 Comparisons of average latencies between 100 Mbps and 10 Mbps network environments	79
4.19 Comparisons of object throughputs between 100 Mbps and 10 Mbps network environments	80
4.20 Comparisons of byte throughputs between 100 Mbps and 10 Mbps network environments	80
4.21 Results from revised server processing model and SURGE measurements $\mathbf{s}=1.143$	81
5.1 The clustered server simulation system architecture	84
5.2 Percentage of bytes transferred (HTTP/1.0, Bryhni-CONN-G, $\mathbf{s}=1.143$, $N_{UE}=640$)	88
5.3 Percentage of bytes transferred (HTTP/1.1, Bryhni-CONN-G, $\mathbf{s}=1.143$, $N_{UE}=640$)	89
5.4 Percentage of bytes transferred (HTTP/1.0, XSIZE-S, $\mathbf{s}=1.143$, $N_{UE}=640$)	89
5.5 Percentage of bytes transferred (HTTP/1.1, XSIZE-S, $\mathbf{s}=1.143$, $N_{UE}=640$)	90
5.6 Average latency comparisons ($\mathbf{s}=1.143$, $N_{UE}=640$)	92
5.7 Object throughput comparisons ($\mathbf{s}=1.143$, $N_{UE}=640$)	93
5.8 Data throughput comparisons ($\mathbf{s}=1.143$, $N_{UE}=640$)	93
5.9 Average latencies for single server and XSIZE-G clustered server ($\mathbf{s}=1.143$)	94
5.10 Object throughputs for single server and XSIZE-G clustered server ($\mathbf{s}=1.143$)	94
5.11 Data throughputs for single server and XSIZE-G clustered server ($\mathbf{s}=1.143$)	95
5.12 Rank of connections $\mathbf{s}=1.143$, $N_{UE}=640$	96
5.13 Percentage of objects transferred $\mathbf{s}=1.143$, $N_{UE}=640$	98
5.14 Percentage of objects transferred $\mathbf{s}=2.5$, $N_{UE}=640$	98
5.15 Percentage of objects served (XSIZE-G, $\mathbf{s}=1.143$, $N_{UE}=1$)	98
5.16 Percentage of bytes transferred ($\mathbf{s}=1.143$, $N_{UE}=640$)	100
5.17 Percentage of bytes transferred ($\mathbf{s}=2.5$, $N_{UE}=640$)	100
5.18 Comparison of average latencies with different \mathbf{s} 's (RR-S)	101
5.19 Comparison of object throughputs with different \mathbf{s} 's (RR-S)	101
5.20 Comparison of data throughputs with different \mathbf{s} 's (RR-S)	102
5.21 Load balancing in term of bytes with different average file sizes (RR-S, $N_{UE}=640$)	102

5.22 Effect of varying the number of servers on the average latency (RR-G, $\mathbf{s}=1.143$)	103
5.23 Effect of varying the number of servers on the object throughput (RR-G, $\mathbf{s}=1.143$)	103
5.24 Effect of varying the number of servers on the data throughput (RR-G, $\mathbf{s}=1.143$)	104
5.25 Average latency ratios R_l (RR-G, $\mathbf{s}=1.143$)	105
5.26 Object throughput ratios R_o (RR-G, $\mathbf{s}=1.143$)	105
5.27 The alternative simulation architecture	106
5.28 Comparison of average latencies under two different network environments (CONN-G, $\mathbf{s}=1.143$)	107
5.29 Comparison of data throughputs under two different network environments (CONN-G, $\mathbf{s}=1.143$)	107
5.30 Comparison of average latencies with different averaging windows (XSIZE-G, $\mathbf{s}=1.143$)	108
5.31 Comparison of object throughputs with different averaging windows (XSIZE-G, $\mathbf{s}=1.143$)	109
5.32 Rank of connections under XSIZE-G policy $\mathbf{s}=1.143, N_{UE} = 640$, averaging window = 0.1s	110
5.33 Percentage of objects served (XSIZE-G, $\mathbf{s}=1.143, N_{UE} = 640$, averaging window = 0.1s)	110
5.34 Comparison of average latencies with different averaging windows (CONN-G, $\mathbf{s}=1.143$)	111
5.35 Comparison of object throughputs with different averaging windows (XSIZE-G, $\mathbf{s}=1.143$)	111
5.36 Comparison of average latencies with different averaging windows (RT-S, $\mathbf{s}=1.143$)	112
5.37 Comparison of object throughputs with different averaging windows (RT-S, $\mathbf{s}=1.143$)	113
5.38 Rank of connections under RT-G policy $\mathbf{s}=1.143, N_{UE} = 640$	113

List of Abbreviations

ACK	Acknowledge
ARP	Address Resolution Protocol
CD	Carrier Detect/Collision Detect
CGI	Common Gateway Interface
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Medium Access
DNS	Domain Name Server
FTP	File Transfer Protocol
GIP	Graphic Interchange Format
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Standards Organizations
ISP	Internet Service Provider
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
LL	Link Layer
MAC	Medium Access Protocol
NAK	Negative Acknowledgement
NNTP	Network News Transfer Protocol
ns	The Network Simulator
OSI	Open Systems Interconnection
PC	Personal Computer

PL	Physical Layer
QoS	Quality of Service
RAM	Random Access Memory
RTC	Round-Trip Correction
RTT	Round-Trip Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network
WWW	World Wide Web

Chapter 1

Introduction

In this era of high technology, an incredible number of Web sites are added to the Internet everyday. As some of these sites become popular, they sometimes experience exponential growth in traffic, which soon exceeds the capacity of any single-server architecture. Not properly handled, this growth can lead to frustrated and dissatisfied users unable to access a site. As a result, service providers have no alternative but to use either large multiprocessor or distributed network servers in order to achieve a reasonable quality of service. The problems experienced by the single server can be solved by migrating the software to a faster, more powerful machine. While this tactic relieves short-term pressures, the upgrading process is complex, and the original machine may be wasted. Many companies facing similar situations find that they must continuously upgrade the servers in order to keep up with the service demand. Furthermore, the server is a single point of failure. “The only viable long term solution would embody some form of load distribution across a multi-server configuration.” [Katz 1994]

A new architecture using clusters of workstations interconnected by a high-performance network is emerging and presents a promising solution for building a high-performance and highly available server. Clusters are well suited to Internet service workloads, as they are highly parallel (many independent simultaneous users). Figure 1.1 presents a typical architecture for a cluster of workstations. Each node is a full-blown workstation or PC, with a processor, multiple levels of caches, main memory, a disk and a network interface. All nodes in the cluster store the same documents or have access to data stored on any disk via a distributed file system. The nodes are interconnected by a high-performance network (such as 100Mbps or Gigabit Ethernet). The network can also connect the nodes to a bridge or router that links the cluster to the Internet depending on the architectures chosen. The Web cluster is publicized with one URL and one Virtual IP address (VIP). This last parameter is the IP address of a centralized load balancer with full control over client requests. The load balancer receives all inbound packets for the VIP address and distributes them among the Web servers based on factors like capacity, availability, response time, current load, historical performance, and administrative weights. This type of loosely coupled architecture is more scalable, more cost-effective and more reliable than the single processor system or the tightly coupled multiprocessor system. Clustering enables transparent growth. In other words, physical servers can be added without externally-visible network changes. Clustering also improves fault-tolerance, i.e. a physical server can be taken down for maintenance or repair without resulting in a service outage.

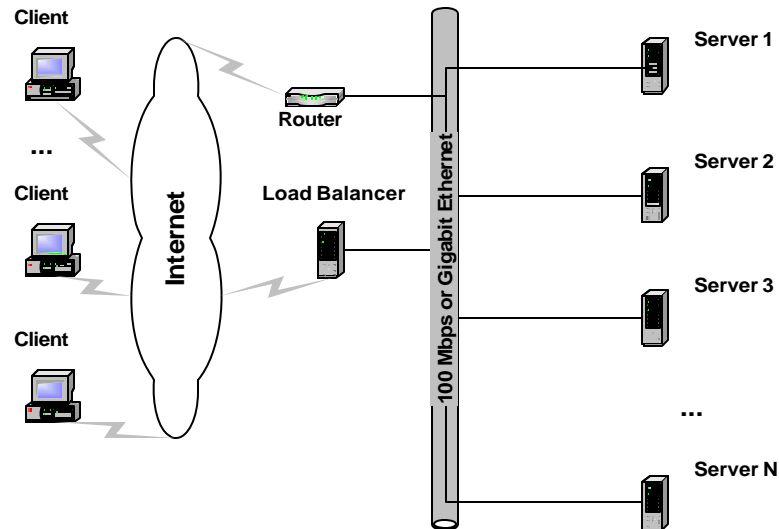


Figure 1.1 Illustration of clustered servers

Server load balancers have become critical infrastructure components for ensuring optimal performance of large Internet or intranet sites and a variety of clustering technologies are now commercially available. However, to our knowledge, no full-range head-to-head comparison of server clustering techniques has ever been made publically available. Most vendors develop their own testing tools to simulate large number of Web users, and most performance tests seem to be done by vendors themselves, with little independent verification. Performance comparisons are difficult due to this lack of standardization of testing tools.

1.1 Thesis Goals

In this thesis, we compare the performances of different clustering technologies through simulation in the *ns2* [Fall 2002] software environment . The simulation repeats real-world experiments and explores possible architectures to be used in the future. Floyd illustrated “Internet simulations are most useful as a tool for building understanding of dynamic, or to illustrate a point, or to explore for unexpected behavior.” [Floyd 2001]

In this study, we will first stress-test a physical Web-server using SURGE (Scalable URL Reference Generator), a load generator that runs on client workstations and which synthetically generates loads that follow the distributional characteristics of the Web traffic. Simulation

models that describe the request/response process of the server will then be developed from the stress-test experiments and the resulting simulations compared with the original measurements for a “reality check”. The server models will then be integrated into the simulation system to mimic different networking environments. Two clustering architectures and five scheduling algorithms will be tested in this manner.

Through a series of simulations in clusters of 2, 4, and 8 servers, we wish to determine the architecture handling more throughputs in a shorter response time, the scheduling algorithms with the lowest response time, and most importantly, the algorithms with the best load balancing. Also we want to study the impact of some important parameters in the implementations of the scheduling algorithms, such as network bandwidth and the interval for polling the servers or monitoring their behavior.

1.2 Summary of Contributions

We view our contributions as follows:

- The creation of server processing models under *ns2* that are verified by stress tests on a real web-server running Apache 1.3.12.
- To model CPU time costs as both a delay and a use of resources.
- To develop an independent simulation tool to test clustered Web servers with consideration of the characteristics of Web workloads.
- To show that measurement in terms of the percentage of objects served is a better indication of load balancing than that of bytes transferred.
- To provide an efficient and inexpensive framework for comparison and evaluation of different architectures and scheduling algorithms for clustering Web servers.

1.3 Thesis Outline

Chapter 2 gives a background overview of Open Systems Interconnection (OSI) Reference Model, TCP/IP network model, Hypertext Transfer protocol (HTTP) and of existing clustered

server architectures and algorithms. A description of the characteristics of Web workload and of the Web Server testing tool, SURGE, is presented in Chapter 3. Chapter 4 describes the stress tests on a Linux server. The results are used to build server processing models in simulation. These server models are then integrated into a simulation system and validated by comparing the simulated results and measurements from SURGE. In Chapter 5, the simulations of clustered server architectures will be described in detail as well as comparisons on the experimental results. Chapter 6 concludes the paper with some suggestions for future work.

Chapter 2

Techniques for Scalable Web Server Clustering

This chapter provides background information on current methods for building scalable web-servers via clustering. The chapter begins with an overview of the Open Systems Interconnection (OSI) Reference Model and the Hypertext Transfer Protocol (HTTP), which form the basis for our simulations. A detailed summary of clustering schemes for web-servers follows this overview along with a review of the associated scheduling algorithms.

2.1 OSI Reference Model and TCP/IP Network Model

The OSI Reference Model [Tanenbaum 1996] is based on a proposal developed by the International Standards Organization (ISO), and defines seven network layers, each one built upon its predecessor: (Figure 2.1)

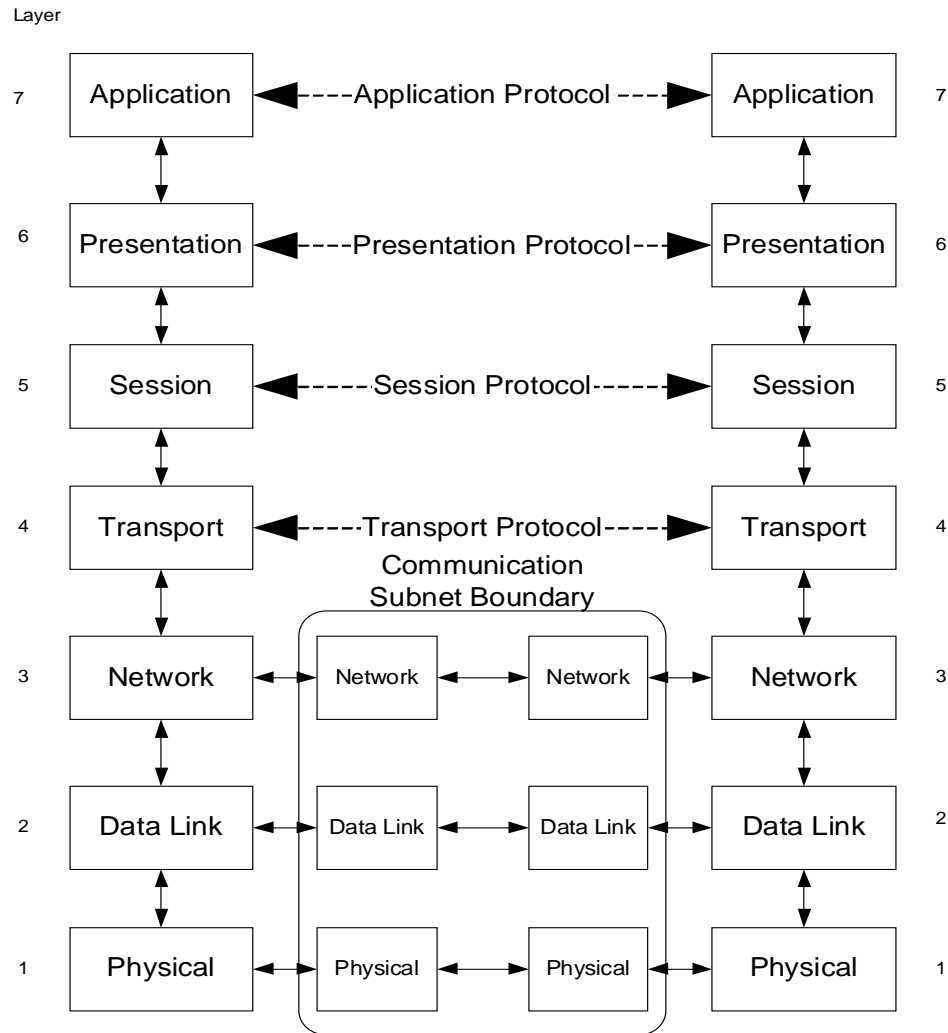


Figure 2.1 Network architecture based on the OSI reference model

1. The **physical layer** is concerned with transmitting raw bits via a communication channel. In this layer, the standard includes the coding and modulation schemes that are used to represent the actual 1s and 0s, as well as low-level details such as the types of connectors.
2. The **data link layer** defines the format of data on the network. The sender breaks the input data up into data frames. A network data frame, also known as packet, includes a checksum, source and destination address, and data. The largest packet that can be sent through a data link layer defines the Maximum Transmission Unit (MTU): it is a fixed upper bound on the amount of data that can be transferred in

one physical frame. Similarly, the network interface MTU determines the packet size limit in the Internet. The data link layer uses a network interface to handle the physical and logical connections to the packet's destination. Our testing network environment is Ethernet LAN. Ethernet networks are used extensively in technical and office environments. A host connected to an Ethernet has an Ethernet interface to handle connections to the outside world, and a loopback interface to send packets to itself. Ethernet has a maximum frame payload of 1500 bytes, so the biggest MTU that can possibly be used on an Ethernet is 1500 bytes. The minimum allowable MTU setting is 576 bytes and an intermediate MTU is about 1000 bytes. The intermediate MTU is used in this study. An Ethernet host is uniquely addressed by its Media Access Control (MAC) address. MAC addresses are usually represented as six colon-separated pairs of hex digitals. This number is unique and is associated with a particular Ethernet device. The data link layer's protocol-specific header specifies the MAC address of the packet's source and destination.

3. The **network layer** provides for the transfer of data between two hosts. This layer is responsible for the establishment, maintenance and termination of the connection between two hosts across any intervening communication facilities. This layer also manages addressing, routing and bottleneck prevention. The Internet Protocol (IP) [Freeman 1995] is commonly used as a network layer interface. The Internet Protocol identifies each host with a 32-bit IP address. IP addresses are written as four dot-separated decimal number between 0 and 255, e.g., 192.168.0.1. Even though IP packets are addressed using IP addresses, hardware (MAC) addresses must be used to actually transport data from one host to another. The Address Resolution Protocol (ARP) is used to map the IP address to its hardware address.
4. The basic function of the **transport layer**, is to accept data from the session layer, split it up into smaller units if needed, pass these units to the network layer, and ensure that all pieces arrive correctly at the other end. Another task is to hide all

the network dependent characteristics from the three layers above it – especially to isolate the session layer from the hardware technology. The two prevalent transport layer protocols used in the Internet are the Transmission Control Protocol (TCP) [Postel 1981] and the User Datagram Protocol (UDP) [Postel 1980]. TCP is fairly heavy “connection-oriented” protocol, while UDP is “light-weight” and connectionless. Additional details on the Internet protocol layers are given in Section 2.1.1. Reliability and speed are the primary differences between these two protocols. TCP establishes connections between hosts on the network through ‘sockets’ which are determined by the IP address and port number. TCP also keeps track of the packet delivery order as well as the packets that must be resent. Maintaining this information for each connection makes TCP a state oriented protocol. This project is built on TCP. TCP provides for reliable network transmission. UDP, on the other hand, is stateless and provides a low overhead transmission service but less error checking. UDP has no ability to perform error or flow control. This simplicity reduces the overhead and makes UDP faster than TCP in performance when the network is not congested; however, under congestion, UDP-based application will most likely result in poor throughput due to many packet losses.

5. The **session layer** allows users on different machines to establish sessions between them. A session is the period of time during which two users remain logically connected. It might be used to allow a user to log into a remote time-sharing system or to transfer a file between two machines. One example is the AppleTalk Data Stream Protocol (ADSP) [Apple Computer 1996] which guarantees reliable data transfer between two nodes.
6. Unlike all the lower layers that exist solely to move bits reliably from one location to another, the **presentation layer** is concerned with the syntax and semantics of the information transmitted. A typical example of a presentation service is encoding data in a standard, agreed upon way. Additionally, this layer may implement text compression techniques or encrypt the data to be transmitted.

7. The **application layer** is the highest layer in the OSI reference model. It provides network services to the end-users. Protocols are provided for functions such as file transfer, electronic mail and Web browsing.

Although the OSI model is widely used and often cited as the standard in layered architectures, the TCP/IP protocol has become extremely popular in the industry. TCP/IP is designed around a simple four-layer scheme and omits some features found in the OSI model. TCP/IP protocol also combines features from adjacent OSI layers. The four network layers defined by TCP/IP model are the **link layer** which defines the network hardware and device drives, the **network layer** responsible for basic communication, addressing and routing, the **transport layer** establishing communication among programs on a network and the **application layer** controlling all end-user applications. Figure 2.2 shows the network architectures based on the OSI model and TCP/IP protocol with the unit of information at each TCP/IP layer. For the rest of this Chapter, layers will refer to OSI model only.

Application Layer	Applications and Services (message)
Presentation Layer	
Session Layer	
Transport Layer	TCP or UDP (segment or datagram)
Network Layer	IP (datagram)
Data Link Layer	Data Link (frame)
Physical Layer	Physical Layer (bit)

Figure 2.2 Network architectures based on the OSI reference model and TCP/IP protocol

2.1.1 The TCP/IP Protocol

The Internet Protocol (IP) is used to route information packets (or datagrams) through a network and uses addresses carried in the header to specify a packet's destination. IP is a connectionless datagram service and known as a best-effort protocol. There is no error control of data, no retransmissions and no flow control. In the Internet, TCP makes up for

IP's lack of reliability through end-to-end confirmation that packets have arrived. TCP is responsible for breaking up the message into chunks called segments. A TCP header, attached to each segment, contains one important piece of information: the sequence number. This number is used upon arrival to ensure the receipt of datagrams in the proper order.

The IP layer is used to transfer data between hosts. To achieve this goal, the protocol adds an IP header to the TCP packet and dispatches it to the destination IP address. Upon arrival, the added header is removed. When no errors are detected, an acknowledgement message is sent to the originating TCP device. Otherwise, TCP throws away the datagram. When the originating TCP device fails to receive the acknowledgement within a certain time period, it resends the datagram. TCP also provides the flow control that controls the rate at which data are exchanged between two hosts. Flow control involves a feedback mechanism that informs the source machine of the destination machine's ability to keep up with the current flow of data transmission. The IP packet data is forwarded to the right application using the port number included in the TCP header. Another important process that IP implements is called fragmentation and reassembly. IP uses a fragmentation process to split datagrams that are larger than the network can handle into smaller fragments (less than the MTU). Once a datagram has been fragmented, it will not be reassembled until it reaches the final destination node. In this way, the higher layer protocols can send datagrams of whatever size they prefer.

2.1.2 Overview of HTTP Protocol

The HyperText Transfer Protocol [W3C 2001] or HTTP is a standard method for transmitting information through the Internet, and is the primary protocol for information exchange. Typical Web pages consist of HyperText Markup Language [W3C 2002] (HTML) text as well as many embedded objects (graphics/video et cetera). These objects are usually independent of each other and can be retrieved separately. In July 1999, the

World Wide Web Consortium (W3C) celebrated the arrival of HTTP/1.1 [Fielding 1997; W3C 1999] as an IETF (The Internet Engineering Task Force) [Force 2002] Draft Standard. Both HTTP/1.0 [Berners-Lee 1996] and HTTP/1.1 use the TCP protocol for data transport. For the purposes of our work, the main difference between HTTP/1.1 and HTTP/1.0 is the inclusion of *persistent connections*; more details on this will be presented shortly.

Figure 2.3 depicts the interaction between HTTP/1.0 clients and servers. In the figure, short vertical arrows show local delays at either the client or the server. Solid arrows show mandatory round-trips resulting from packet exchanges. Gray arrows show packets required by the TCP protocol. These packets do not affect the latency because the receiver does not have to wait for them before proceeding. On the left of the graphic, RTT stands for the Round Trip Time required by the HTTP protocol to complete the function stated. The “Finish (FIN)” signal is used to close a session gracefully. We now present a step-by-step view of the communications process:

1. The client browser opens a TCP/IP connection with the server, resulting in an exchange of SYN packets in the initial messages when setting up a data transfer as part of TCP’s three-way handshake procedure. The client then sends a connection request (SYN), the server responds (SYN), and the client acknowledges the response with an ACK flag.
2. The client transmits an HTTP request via a DAT package to the server. The server reads from its memory or disk, and sends a response message containing the page’s main body, in the HTML language. The server then closes the connection.

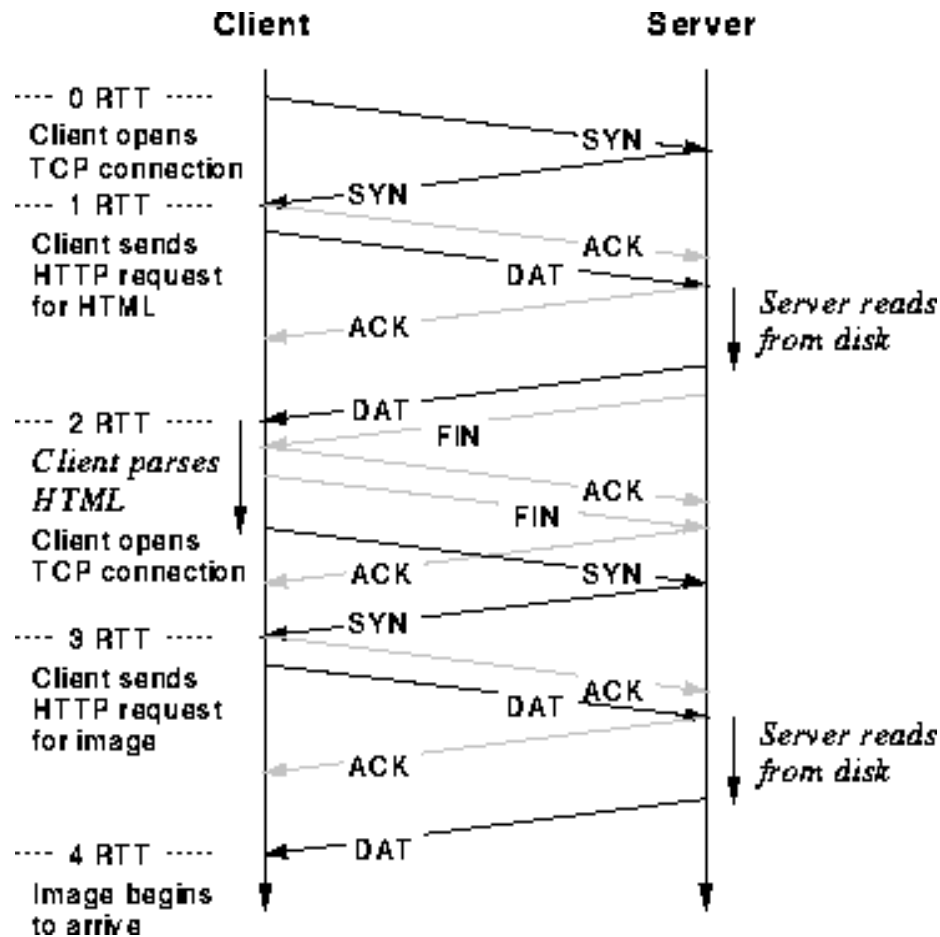


Figure 2.3 Packet exchanges and round-trip times for HTTP/1.1 [Rubarth-Lay 1996]

- After parsing the returned HTML document to extract the URLs for embedded images, the client opens up to a maximum of N connections with the server. At this point, the client transmits HTTP requests for the inlined images. Parallel HTTP/1.0 requests could be used to reduce the latency.
- For each connection, the server responds with the image file it reads from the disk, and then closes the connections.

5. The cycles of request-response (Step 3-4) continue until all of the inlined images are withdrawn.

Instead of closing the connection in Step 2 and 4 as in HTTP/1.0, HTTP/1.1 leaves the TCP connection open between consecutive request/response operations. This “persistent-connections” allows the retrieval of multiple objects in one connection, saving the object transfers from going through the TCP slow-start phase at the cost of use of higher main memory. Persistent connections have been found, on average, to reduce both the overall network traffic and the page response time and we thus use this feature in our study. However, if we combine persistent connections with the parallel requests discussed in Step 3 above, then it is possible to have multiple parallel persistent connections, which could be harmful. Liu [1999] showed that the response time increases exponentially with the number of persistent parallel connections. In the two most popular browsers: Netscape seems to use 4 parallel persistent connections, and Internet Explore (IE) seems to use 6 [Nielsen 1997], in this study the default is $N = 4$.

2.2 Cluster-Based Scalable Service Architectures

Server clustering refers to the pooling together of many servers to act as a single unit. The technique provides incremental scalability, allowing the reuse of seemingly obsolete office or classroom systems in Web-servers. The following is a review of Web server clustering architectures.

2.2.1 Server Selection Strategies

There are a number of server selection strategies for selecting a server from a cluster to service a user request. The strategies can be broadly divided into five categories: client-side approaches, server-side HTTP redirect, clusterings via network address translation, IP tunneling and IP modification.

1. The Client-Side Approach

In the client-side approach, an applet is operating from the client's side. The applet makes requests to the cluster of servers to collect the information such as load, geographical distance [Seltzer 1995], measured latency [Chankhunthod 1996] and bandwidth [Carter 1996] from all the servers, chooses a server based on that information and forwards requests to that server. For example, server clusters providing multimedia content could offer clients a choice of content fidelity with different estimates of transfer rates. The client-side approach is appropriate when the group of servers is heterogeneous or widely dispersed across the network. This approach is not transparent to the client. Furthermore, the method requires the modification of client applications, implying that one modification does not apply to all TCP/IP services. The client-side approach increases network traffic through the generation of extra queries or probes.

Products: Berkeley's Smart Client [Yoshikawa 1997]; Harvest [Chankhunthod 1996]

2. The Server-Side HTTP Redirect

This simple approach uses the HTTP "redirect" command to route incoming users to one of a number of available servers [Andresen, Yang, Holmedahl 1996; Delgadillo 2000]. As illustrated in Figure 2.4, the client makes an initial connection to the main server using the publicized URL. The server ([www.javelin](http://www.javelin.com)) responds with a redirect instruction to one of many available hosts, and the client then makes all subsequent requests to that host. In this approach, server clusters typically contain members with similar resources and a shared local network. "URL Redirection" has the advantage of being a standard part of all Web servers. This mechanism, however, makes visible the URLs of all server machines to the clients, which can potentially be stored in favorite lists, be indexed by search engines, and so on. Those situations would defeat subsequent load balancing.

In this approach, the main server experiences unnecessary overhead from its redirection role. The establishment of two TCP connections also generates an increase in network traffic.

Products: SWEB [Andresen, Yang, Holmedahl 1996]; Cisco's DistributedDirector [Delgadillo 2000]

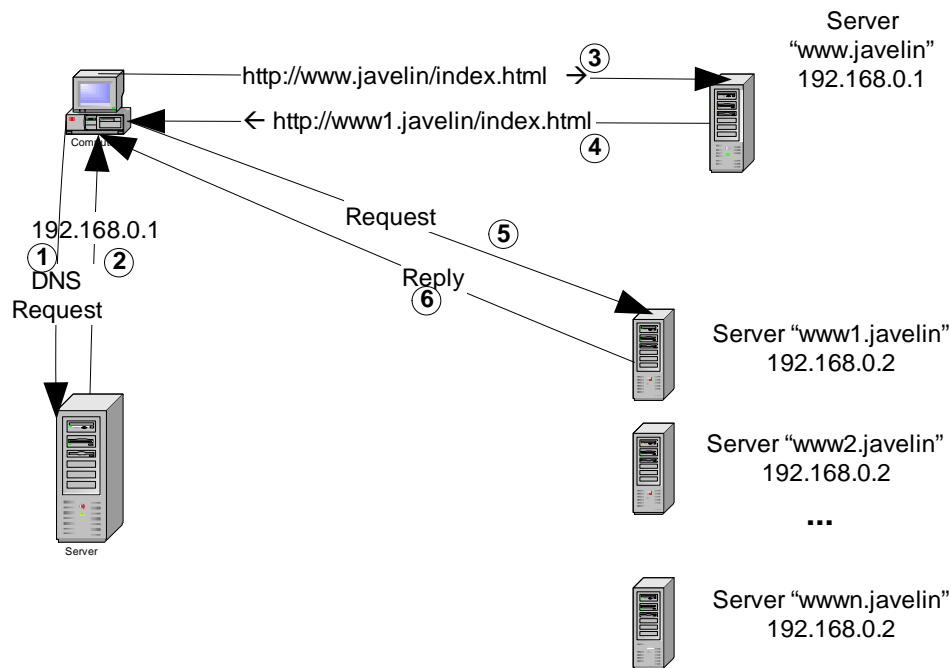


Figure 2.4 The server-side URL Redirection

3. Clustering via Network Address Translation (NAT)

Clustering via Network Address Translation (NAT) [Egevang 1994] is also referred to as Layer Four Switching with Layer Three Packet Forwarding [Schroeder 2000] or L4/3 clustering. The impetus towards increasing use of NAT comes from a number of factors:

- A world shortage of IP addresses in IP_{v4} (version 4)
- Security needs
- Ease and flexibility of network administration

The need for NAT arises when hosts in the internal network want to access the Internet and also be accessed by the public Internet. In order to achieve this goal, the load balancer substitutes all private IP addresses by a globally registered IP address for all messages leaving the private network. All private IP addresses are restored by the load balancer in reply messages entering the private network. During the translation, any integrity codes affected – such as packet checksums, cyclic redundancy checks (CRCs), or error correction checks (ECCs) – are recomputed.

The architecture of clustering via NAT approach is illustrated in Figure 2.5. The load balancer and Web servers are interconnected by a switch or a hub. The servers usually all provide the same service. The data contents of these servers are either replicated on each of their local disk, shared on a network file system, or served by a distributed file system.

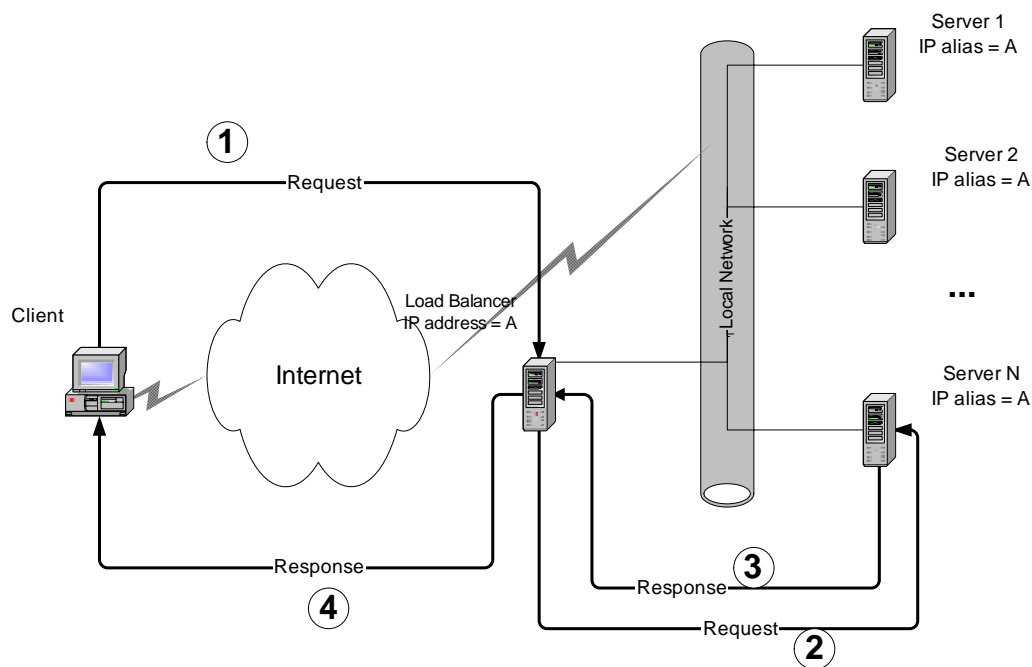


Figure 2.5 Clustering via NAT

The traffic flow in a NAT approach is summarized as follows:

- A client sends an HTTP packet using the external IP address of the load balancer as the destination.
- The load balancer matches the packet's destination address and port number to those of a Web server service according to the Web server rule table. A Web server is then chosen from the cluster by a scheduling algorithm and the connection is added into the binding table that records all established connections.
- The destination address and the port of the packet are rewritten to those of the chosen server. IP and TCP checksums are recalculated and the packet is forwarded to the chosen server. The packet is rewritten and forwarded to the chosen server when the load balancer determines that the incoming packet belongs to this connection (it is a kernel-level function or device driver which examines only the header of each packet and decides whether the packet belongs to an existing connection, or represents a new connection request. A simple connection table stored in memory is used to achieve this. [Gage 2000]) and that the established connection can be found in the binding table.
- The chosen Web server accepts the packet and replies to the client via the load balancer, which it uses as a gateway. When the reply packets come back from the chosen Web server, the load balancer rewrites the source address and port of the packets to those of the load balancer, recalculates the IP and TCP checksums and sends the packet to the client.
- When the connection terminates or times out, the connection record is removed from the binding table.
- In the event that the packet does not correspond to an established connection and is not a connection initiation packet itself, it is dropped.

Figure 2.6 gives an example illustrating the rules for clustering via NAT: the client (IP:199.160.6.18) makes a request to load balancer (IP: 142.58.142.220) for Web service. Once receiving the request, the load balancer chooses server station 192.168.0.1, rewrites

the packet's destination to 192.168.0.1 and forward the packet. Server 192.168.0.1 sends back the response to the load balancer by reversing the source and destination addresses of the incoming request. Receiving the response, the load balancer changes the source address to its own IP and sends the packet to the client.

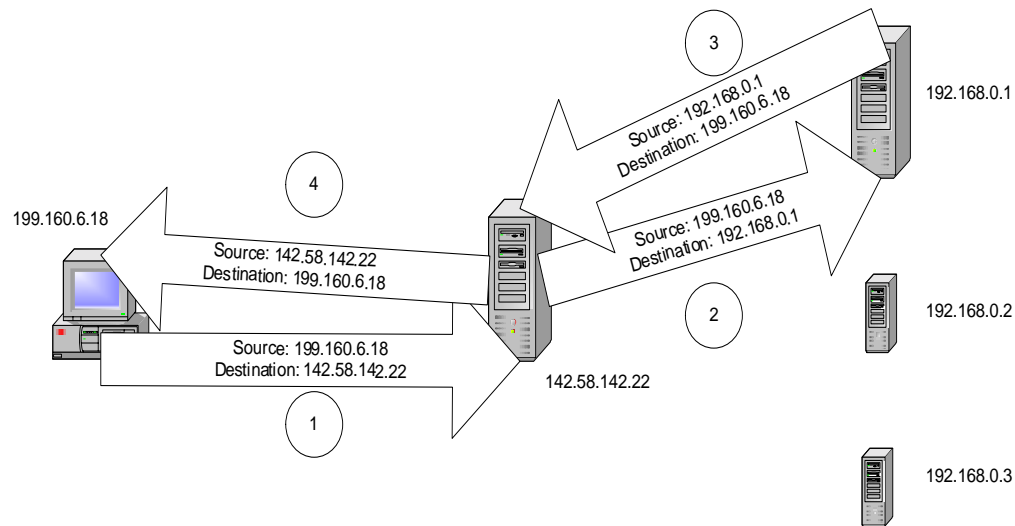


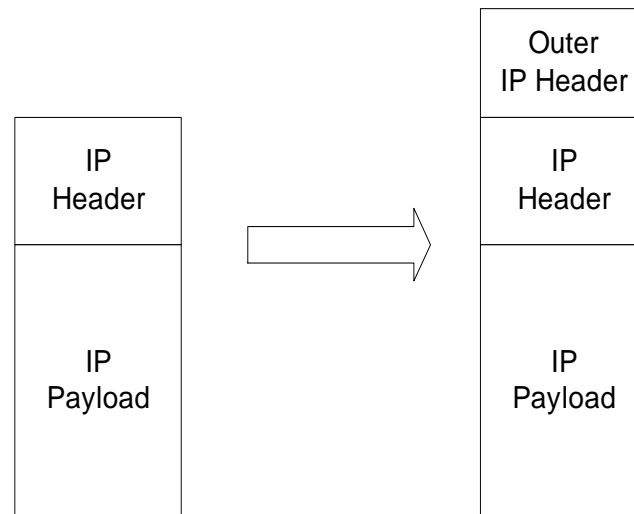
Figure 2.6 Illustration of clustering via NAT

Products: LSNAT [Srisuresh 1998]; Berkeley's MagicRouter [Anderson 1996]; Cisco's LocalDirector [Cisco Systems 2001]

4. Clustering via IP Tunneling

IP tunneling [Perkins 1996] is a technique to encapsulate IP datagrams within other IP datagrams in order to redirect them to another IP address.

“To encapsulate an IP datagram using IP in IP encapsulation, an outer IP header is inserted before the datagram's existing IP header, as follows:



The outer IP header Source Address and Destination Address identify the "endpoints" of the "tunnel". The inner IP header Source Address and Destination Address identify the original sender and recipient of the datagram, respectively. The inner IP header is not changed by the encapsulator, except to decrement the time-to-live (TTL) value, and remains unchanged during its delivery to the tunnel exit point. No change to IP options in the inner header occurs during delivery of the encapsulated datagram through the tunnel. If need be, other protocol headers such as the IP Authentication header may be inserted between the outer IP header and the inner IP header." [Perkins 1996]

When this technique is used to implement server clustering, the load balancer tunnels the request packets to the different servers. The servers then process the requests and return the results directly to the clients. In order to accomplish this, the secondary address¹ of each Web server is configured to be the same as the primary address of the load balancer. This configuration is done through the use of interface aliasing, which is the mechanism

¹ This is called multihoming. A secondary IP address can be configured on the same interface that has the primary IP address.

that enables a single physical or virtual network port to assume responsibility for more than one IP address, or by setting the alias on the loopback adapter [Gage 2000].

The architecture of clustering servers via IP tunneling is illustrated in Figure 2.7.

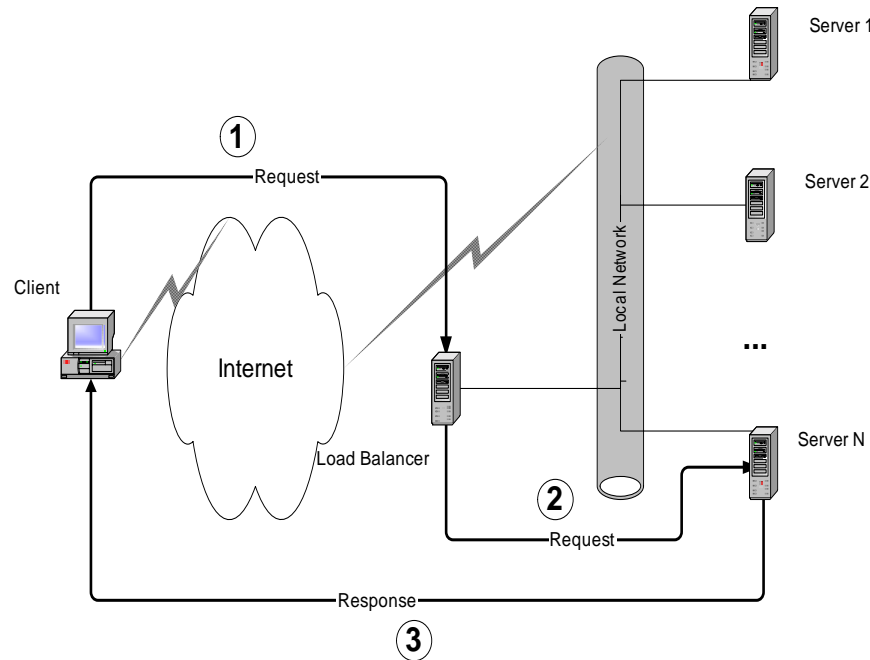


Figure 2.7 Clustering via IP tunneling or IP Modification

The traffic flow of clustering via IP tunneling is as follows:

- A client sends an HTTP packet using the external IP address of the load balancer as the destination.
- The load balancer matches the packet's destination address and port number to those of a Web server service according to Web server rule table. A Web server is then chosen from the cluster by a scheduling algorithm and the connection is added to the binding table that records all established connections.
- The load balancer encapsulates the packet within an IP datagram and forwards it to the chosen server. The packet is again encapsulated and forwarded to the server

when the load balancer determines that the incoming packet belongs to this connection and that the chosen server can be found in the binding table.

- Upon receipt, the server decapsulates the packet, processes the request and returns the result to the client directly.
- When the connection terminates or timeouts, the connection record is removed from the binding table.
- In the event that the packet does not correspond to an established connection and is not a connection initiation packet itself, it is dropped.

The main difference between the use of IP tunneling and that of NAT in clustering architecture is that the load balancer sends requests to Web servers through IP tunnels in the former, and via network address translation in the latter. The IP tunneling architecture avoids the rewriting of packets in the transport of information from servers to clients, which is particularly important when serving large volumes of data. The request packets only transmit URL addresses and are typically short. On the other hand, reply packets potentially carry large amount of data as they may be comprised of images and audio/video etc. Another important difference is that each constituent server can be configured with a private IP address in the case of NAT, while all the Web servers in IP tunneling must be configured with a globally unique IP address.

Products: Linux Virtual Servers [Zhang 1999]

5. Clustering via IP Modification

Clustering via IP modification is also called Layer Four Switching with Layer Two Packet Forwarding [Schroeder 2000] or L4/2 clustering. In L4/2 clustering, a load balancer does not modify the client's IP packet when forwarding it. This technique assumes that the load balancer is on the same subnet as the Web servers, all the Web servers and the load balancer share the same IP address. The load balancer simply forwards the packet explicitly to the chosen server using its physical address (MAC

address) on the LAN without modifying the IP header [Cardellini 1999]. Upon the receipt of the request, the chosen server's TCP stack establishes the server-to-client part of the connection according to standard TCP semantics. This task is accomplished by swapping the source and the target addresses as supplied by the client rather than determining them from its own basic configuration. The result is that the server replies to the client using the load balancer's IP address. Provided that the standard TCP/IP protocols are implemented, the load balancer is not dependent upon server platform.

The traffic flow of clustering via IP modification is as follows:

- A client sends an HTTP packet using the external IP address of the load balancer as the destination.
- The load balancer matches the packet's destination address and port number to those of a server service according to Web server rule table. A Web server is then chosen from the cluster by a scheduling algorithm and the connection is added to the binding table that records all established connections.
- The load balancer's TCP/IP stack re-addresses the MAC frame containing the packet and the packet is then forwarded to the chosen server. The load balancer forwards the packet to the server when the load balancer determines that the incoming packet belongs to this connection and that the chosen server can be found in the binding table.
- The server accepts the packet, swaps the source and the target addresses as supplied by the client and returns the result to the client directly.
- When the connection terminates or times out, the connection record is removed from the binding table.
- In the event that the packet does not correspond to an established connection and is not a connection initiation packet itself, it is dropped.

Products: IBM Network Dispatcher [Gage 2000]; ONE-IP [Damani 1997]; LSMAC [Gan 2000]; Alteon ACEDirector [Networks 2002]

2.2.2 Advantages and Disadvantages

Clustering via NAT (L4/3 Clustering)

The NAT approach has several advantages. First, Web servers can run any operating system that supports the TCP/IP protocol. Second, real servers can use private Internet addresses and thus only one public IP address is needed.

NAT's major disadvantage is its poor scalability. Since both request and reply packets need rewriting by the load balancer, it may become a system bottleneck when the size of the Web servers' throughput exceeds the capacity of the load balancer. To address this problem, a hybrid approach can be applied. The hybrid method will be discussed in detail towards the end of this chapter.

Clustering via IP Tunneling

One key performance and scalability benefit of the clustering via IP tunneling is that the application server responds to the request directly to the client without travelling back through the load balancer. The reply does not need the original path and can use a separate high-bandwidth connection. Incoming packets are short and reply packets are large in most Web services, enabling the load balancer of the cluster to handle many requests. Therefore, IP tunneling greatly enhance the load balancer's capacity to handle a large number of web servers.

Clustering via IP Modification (L4/2 Clustering)

Besides the advantages of clustering via IP tunneling, IP modification is transparent to both the client and the server because it does not require packet rewriting. On the other hand, IP tunneling is only transparent to the client, since it is not aware that its requests are handled by a hidden Web server. IP tunneling architectures require modifications to the kernel code of the server, since the encapsulation occurs at the TCP/IP level.

[Cardellini 1999]

2.3 Scheduling Algorithms

Scheduling algorithms can be categorized into four main classes: stateless, statistical, dynamic and content-based. In clusters, the primary concern is balancing the request load across servers so that when resources are equal a client would see similar response times from all servers. In all cases, the response time depends upon both the resource capacity and current load. Stateless scheduling does not account for resource capacity or availability, but it is very simple to implement. Some examples are Round-Robin DNS and random scheduling. Statistical schedules are computed from past performance data such as latencies and bandwidths and reflect typical levels of contention for the server/network connection. When data exhibit high variability, these schedulers become less reliable. Probes provide an estimate of the current resource availability but do not include all performance factors. Content-based scheduling is gaining popularity in the industry and a great deal of research is currently on-going in this area [Schroeder 2000]. This method supports the delivery of a variety of content types such as images and video/audio, as well as advanced service such as streaming video/audio on demand, content personalization, and dynamic application delivery. We now look at some specific strategies:

- **Round-Robin Domain Name Service (RR-DNS)**

Domain Name Services (DNS) map domain names, also called hostnames, to IP addresses. RR-DNS is the simplest scheme for implementing load-balancing and is performed only once at the beginning of the transaction [Katz 1994]. RR-DNS treats all Web servers as equals regardless of connections, response times or processing capacities [Brisco 1995]. In this scheme, a hostname is mapped to one of several IP addresses, chosen in a round robin manner (Figure 2.8). This system could result in different IP addresses for two consecutive requests. Some advantages of the RR-DNS are protocol compliance, transparency to the client and destination hosts, and standard function shipment on most name-servers. The RR-DNS technique

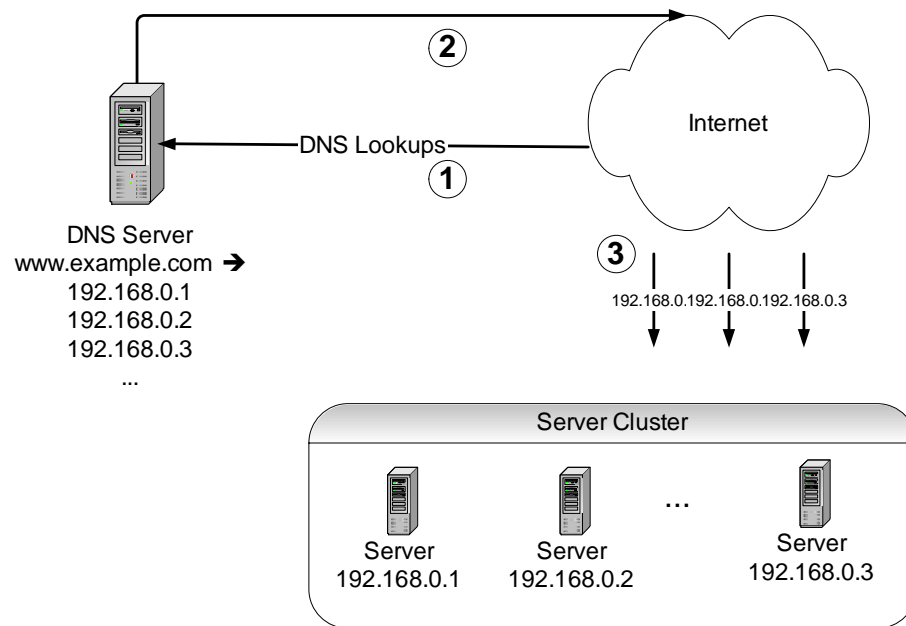


Figure 2.8 Round Robin DNS

is effective when HTTP requests are targeted to HTML information of relatively-uniform size chunks and when the load and computing powers of workstations are relatively comparable. Our results show that RR-DNS can achieve a performance level second only to the dynamic scheme. However, the DNS mapping may be cached by intermediate name servers and possibly clients. Map caching decreases the resolver traffic and therefore speeds up the resolving process. The caching is controlled by a time-to-live (TTL) value appended by the DNS server to each piece of information and specifies a validity period for caching the IP address mapping. After the expiration of the TTL, the address mapping request is forwarded to the cluster DNS for a assignment to a Web server. The TTL value decreases the DNS traffic if set too high, but also causes the intermediate DNS servers or the clients to cache the mapping information too long, leading to poor HTTP traffic distribution over the Web cluster. On the other hand, a small TTL value dramatically increases DNS traffic and the request processing time for visitors, as intermediate DNS servers or clients expire the

mapping information faster, and therefore resolve it more often; this, however, results in better balancing of HTTP traffic.

Products: The NCSA scalable Web server is the first prototype of a scalable clustered server by using the RR-DNS approach. [Katz 1994]; Cisco's LocalDirector [Cisco Systems 2001]; Berkeley's MagicRouter [Anderson 1996]; Alteon ACEDirector [Networks 2002]

- **Weighted Round-Robin DNS (WRR-DNS)**

Weighted Round-Robin scheduling [Zhang 1999] can manage Web servers with different processing capacities. Each server is assigned a weight that indicates its processing capacity. For example, if the three Web servers A, B and C have the weights 4, 3 and 2 respectively, WRR-DNS sequence might be ABCABCABA. As in RR-DNS, workload is not considered in WRR-DDS, which will cause dynamic load imbalance among Web servers when the request loads vary highly.

Products: Cisco's LocalDirector [Cisco Systems 2001]; IBM Network Dispatcher [Gage 2000]

- **Random Scheduling**

In Random scheduling, the load balancer acts as a domain name service server [Katz 1994]. When a client requests the mapping of a domain name, the load balancer selects a Web server at random from the cluster [Anderson 1996] [Delgadillo 2000]. Again, random scheduling takes no account of the different processing capacities or workloads that leads to inefficiencies.

Products: The CIP TN 3270 from Cisco Systems [Cisio 1997]; Berkeley's MagicRouter [Anderson 1996]

- **Least-Connection Scheduling**

In the Least-Connection scheduling algorithm, the load balancer chooses the server with the least number of active connections [Srisuresh 1998; Zhang; Bryhni 2000]. When there is a collection of servers with similar performance, the Least-Connection scheduling is proficient at evening out request distribution. This result is not achieved when servers' processing capacities are highly variable, however. Weighted Least-Connection scheduling [Srisuresh 1998; Zhang 1999] avoids the above problem as servers with higher capacity will be given a larger percentage of the active connections. In this scheme, Web server administrators assign a weight to each Web server. The load balancer divides the number of connections of a server by its weight to calculate server's number identifier. The server with the lowest number identifier is chosen.

Another variation is probabilistic, whereby the probability of selecting a server is inversely proportional to the load on the server [Aversa 2000]. The advantage of this approach is that it avoids possible oscillations (whereby all requests in a short timeframe are re-routed to the server with the lowest advertised load, potentially overloading such a server.).

Least-Connection scheduling does not take into account the fact that some connections are more resource consuming and require more of a server's memory or a longer time to read from disk than the others. The same number of connections does not necessarily lead to same response time.

Products: Cisco's LocalDirector [Cisco Systems 2001]; Alteon ACEDirector [Networks 2002]

- **Least Round-Trip Scheduling**

Least Round-Trip scheduling monitors the request/response phase of each connection through the TCP protocol [Bryhni 2000]. The load balancer calculates the elapsed-

time between the forwarding of a request and its response from the Web server. The load balancer chooses the server with the smallest mean elapsed time during an averaging window. A sliding averaging window is used for smoother results. When the mean elapsed time of several Web servers is the same in the current averaging window, two methods of selection can be used: choosing the one with the least active connection or randomizing.

- **Least Transmitted Size Scheduling**

Another algorithm to track the system load is to measure network load by tracking packet count or byte count leaving from each of the Web servers over a period of time [Srisuresh 1998] [Bryhni 2000]. The server with the smallest count is chosen. In case two or more servers have transmitted the same number of bytes, it chooses the one with the least number of active connections. A sliding averaging window can be used to avoid the discontinuity of average windows.

- **Hybrid Scheduling**

The Hybrid scheduling approach [Cardellini 2000; Gage 2000] (Figure 2.9), uses many load balancers each with its own server cluster and scheduling algorithm. Load balancers are grouped in a single domain name using a method such as RR-DNS or a “Least-Loaded” algorithm. This scheme spreads the scheduling among load balancers that individually might have bottlenecked the Web service. But it adds extra overheads in traffic and elapsed time.

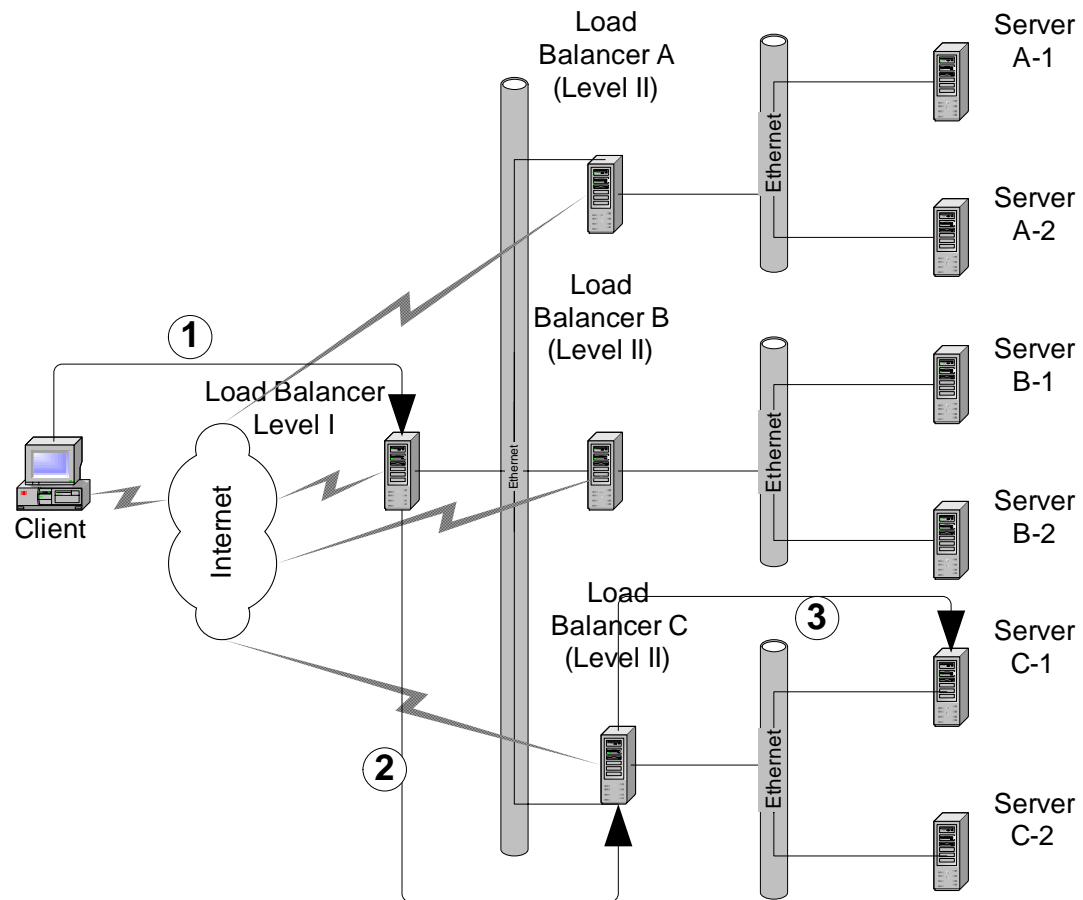


Figure 2.9 Hybrid Scheduling

- **Dynamic or Run-Time Scheduling**




Dynamic or run-time schedulers use small probes to detect current network and/or server conditions [Bryhni 2000] [Cisco Systems 2001]. A simple scheme with limited accuracy is to use the Internet Control Message protocol (ICMP) echo request to regularly poll the different servers [Bryhni 2000]. An accurate response time of each request can be used as a simple load indication. The first Web server to respond is chosen by the load balancer. Probes provide an estimate of current resource availability, but do not include all performance factors. For example, a “ping” probe measures network latency but does not measure the delay taken by the server to read

information from the disk. Another problem arises when conditions fluctuate more rapidly than the time required for the document transfer. Regular requests can coincide with regular processing activity on the monitored server. In other words, the conditions measured by the dynamic probe will not extend over the lifetime of the transfer. Probes can also add run-time overhead in traffic and elapsed time.

Products: Cisco's LocalDirector [Cisco Systems 2001]

- **Content-based Scheduling (L7 Clustering)**

Layer Seven Switching can use Layer Two Packet Forwarding (L2/2) or Layer Three Packet Forwarding (L3/3) [Schroeder 2000]. In other words, this approach uses information contained in OSI Layer Seven (application layer) to augment L2/2 or L3/3 clustering.

Since L7 clustering operates on the content of the client requests, it is also known as Content-Based clustering. Each Web server is allocated a content to serve. Figure 2.10 presents an overview of this process. "Server 1" is capable of handling requests of type , while "Server 2" can handle requests of type  and . We see in the figure that the load balancer decomposes the incoming traffic into two streams, one for server #1 and one for server #2 based on the types of the requests. The traffic flow in Content-Based scheduling is summarized as follows:

- A client sends HTTP requests with the cluster IP as the destination.
- As request arrives from clients, the load balancer accepts the connection as well as the request itself.
- The load balancer inspects the HTTP request content, classifies the requested document into text, audio/video or dynamic and dispatches the request to the appropriate server. The chosen Web server is informed of the status of the client-

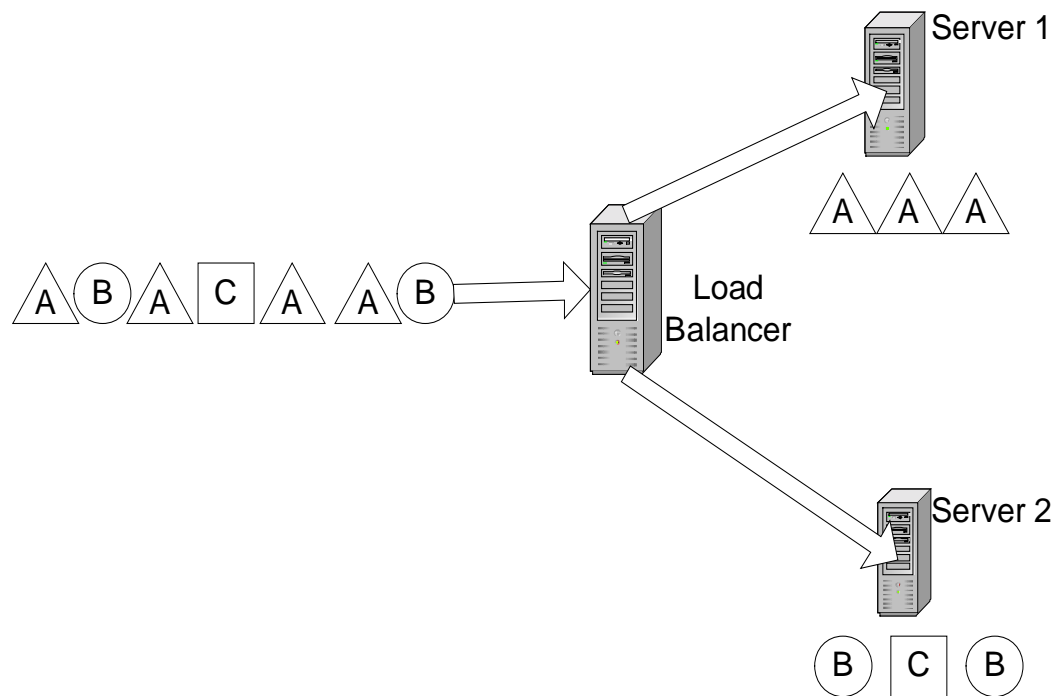


Figure 2.10 Content-based Scheduling [Schroeder 2000]

load balancer connection and takes over that connection; i.e. it communicates directly with the client.

The load balancer keeps track of the status of connections, this information is used to make load-balancing decisions. L7 clustering allows each server's file system to cache data according to its content, allowing higher cache hit-rates to be achieved. Additionally, special servers can handle dynamically generated content, while static requests may be dispatched to servers with less processing power. The overhead of dealing with HTTP requests and replies in the application-layer is high. As such, application-layer load balancers may become system bottlenecks limiting cluster scalability. Similar to the Layer-4 approach, Layer-7 Web switch architectures can be classified by their mechanisms for redirecting packets as well as for the route of the return message.

L7 clustering is limited by the complexity of the content-based routing algorithms [Schroeder 2000]. Content-Based scheduling does, however, make optimum use of the clustered Web servers by caching the Web contents and matching content types with the server capacities according to the resource-consuming level of the content. L7 clustering should provide a higher performance for a given number of server stations than L4/2 or L4/3 clustering alone [Schroeder 2000].

Products: EDDIE [Dahlin 1998]; Reverse-proxy [Engelschall 1998]; SWEB [Andresen]; LARD [Pai 1998]; IBM's Web Accelerator [Levy-Abegnoli 1999]; Cisco's CSS 11800 Content Services Switch [Group 2000]; Web Server Director Pro [Radware]; Zeus Load balancer [Zeus]

- **Master-Slave Scheme**

The Master-Slave scheme [Andresen, Yang, Egecioglu 1996; Iyengar 1997; Zhu 1998] is built on top of Content-Based scheduling. The architecture organizes server nodes into two levels. The master level accepts and processes both dynamic and static content requests. The slave level is used only to process dynamic content upon receiving requests from the master level (Figure 2.11). As dynamic content² generation is more resource-consuming and places greater I/O and CPU demands on the server, the server bottleneck limits the ability of such servers to process large numbers of simultaneous requests. The master-slave scheme has better throughput

² Dynamic data involves information construction at the server before users' requests are answered. Dynamic content enables services such as electronic commerce, database searching, personalized information presentation... To create dynamic content in response to an HTTP request, most servers implement the Common Gateway Interface (CGI) [Team 1998] NCSA HTTPd Development Team (1998). The CGI Specification. <http://hoohoo.ncsa.uiuc.edu/cgi/interface.html> or JAVA. [Microsystems 2002] SUN Microsystems (2002). The source for Java Technology. <http://www.javasoft.com/Common> applications of CGI or JAVA include search engines, hit counters, and redirection.

performance compared to an architecture where both types of contents are served by every node.

Products: Web searching servers at Inktomi [INKTOMI]; AltaVista. [AltaVista]; The Alexandria Digital Library System [Andresen, Yang, Egecioglu 1996]; IBM's Atlanta Olympics web Server [Iyengar 1997]

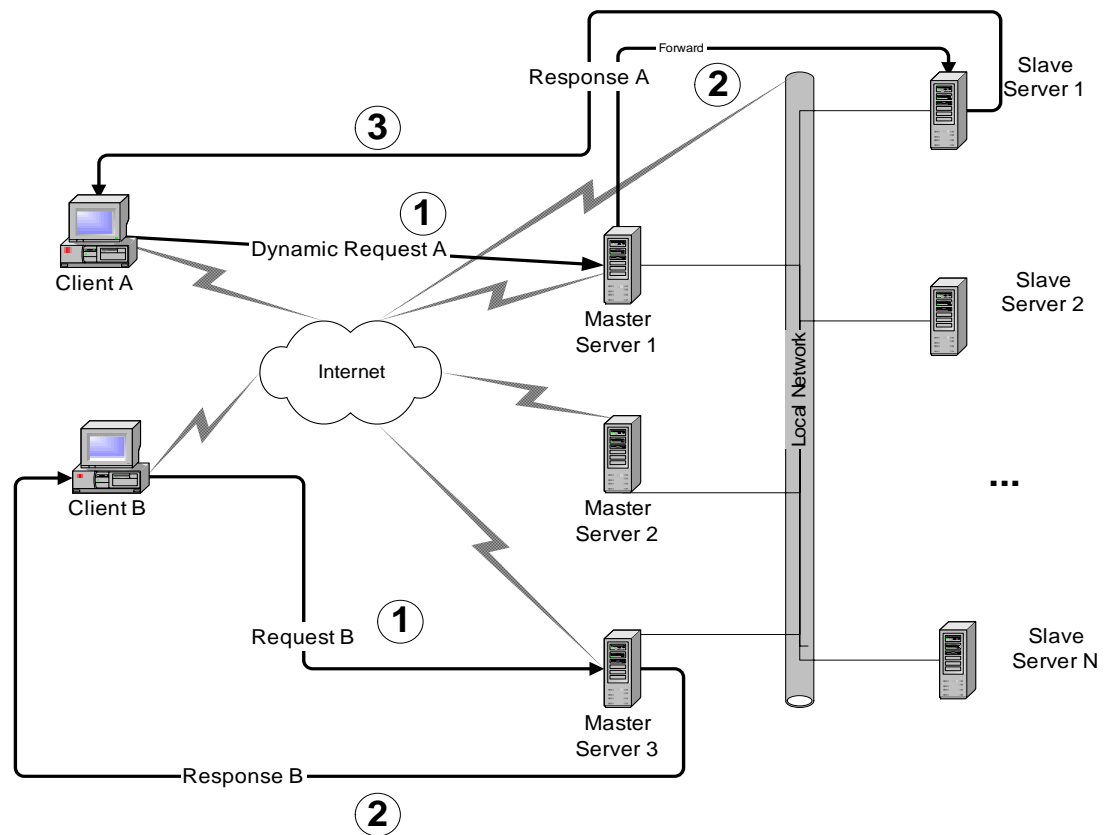


Figure 2.11 Master-Slave Scheme

2.4 Distributed Packet Rewriting

The previously discussed methods for connection routing have employed a centralized node (TCP router) that acts as a load balancer and directs incoming requests to backend hosts. Under these architectures, a single machine whose IP address is published through DNS is responsible for balancing the load across the cluster. This centralized approach is

not inherently scalable, since it does not take into account that the TCP router will be a bottleneck under high load. An alternative to a centralized approach is Distributed Packet Rewriting (DPR). [Bestavros 1998; 1998; Cardellini 2000; Cardellini 2001] DPR distributes requests across many Web servers as TCP routing does (Figure 2.12). The major difference between DPR and TCP routing lies in the publication of IP addresses. DPR uses RR-DNS to publish individual addresses of all machines in the cluster. The responsibility of re-routing requests is therefore distributed amongst all machines. This distributed approach promises better scalability and fault-tolerance than centralized special-purpose connection routers. Aversa et al. [2000] have showed that state-based approaches achieve better throughput and faster response times than stateless methods such as RR-DNS provided that the load estimation is accurate. Our simulation results in Chapter 5 confirm this finding.

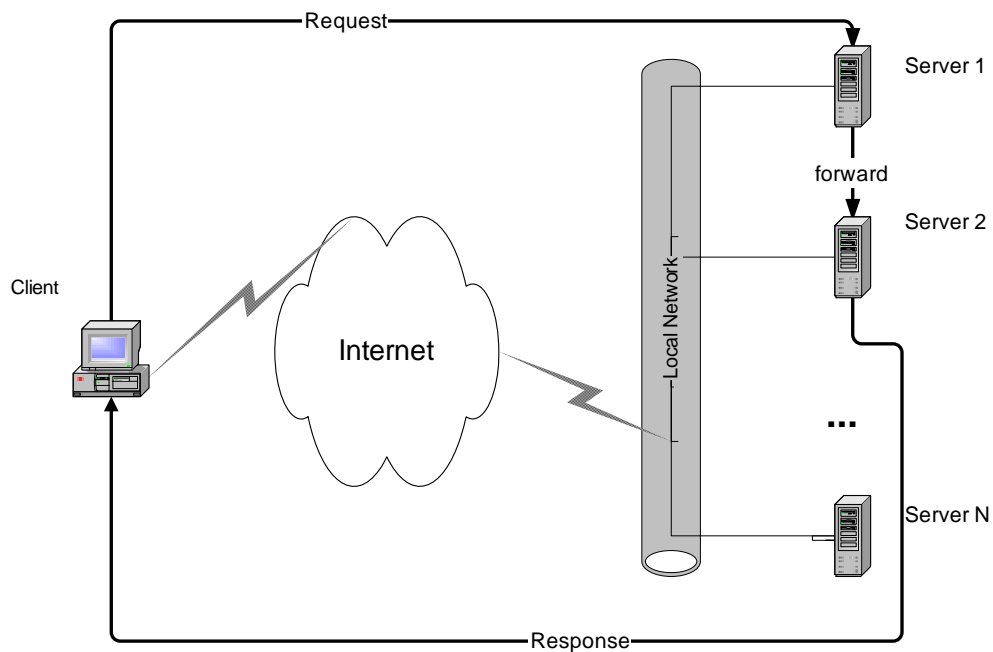


Figure 2.12 Distributed packet rewriting

2.5 Summary

At the beginning of this chapter, we studied the OSI reference model, TCP/IP network model and HTTP protocol. Then we described the cluster-based scalable service architectures and the scheduling algorithms. Later in Chapter 5, we will study some most widely used architectures and scheduling schemes in simulation. We will focus on five commonly used scheduling schemes in the study: Round-Robin, Random, Least-Connection, Least Round-Trip and Least Transmitted-Size. These scheduling algorithms are representative of three major classes of algorithms. The Round-Robin and Random schemes distribute requests independent of the load on each Web server, the Least-Connection approach uses the connection state information for load distributions, while the Least Round-Trip and Least Transmitted-Size policies are based on history statistics of loads.

Chapter 3 will now present the techniques used to determine the performance of Web servers.

Chapter 3

Traffic Model and Performance

Evaluation of Web Servers

In this chapter, we introduce the parameters used to evaluate Web server performance, the distributional characteristics of the Web traffic, and several Web benchmark tools with the emphasis on SURGE (Scalable URL Reference Generator).

3.1 Parameters Used to Evaluate Web Server Performance

The performance of a Web server is dependent upon a number of variables, including the server processing power, disk storage hardware, operating system as well as the network protocols, hardware, bandwidth and congestion. The user perception of a server's performance depends on the client platform and operating environment; and the Web client. The four most common metrics for measuring the capacity of Web servers are:

1. **Requests served per second:** a measure of the maximum number of HTTP requests manageable by a server in a given length of time. A higher number indicates better performance and therefore better service. When a server cannot adequately handle traffic requests, such as during peak periods, some requests will

not be satisfied and result in timeout or “connection refused” messages for the user.

2. **Throughput:** a measure of the maximum amount of data the server can service during a given amount of time. It is measured in bytes per second. Aside from the server’s processing capacity, the network bandwidth and congestion can also influence the perceived throughput of a server.
3. **Round-trip time (RTT):** a measure of the time taken for a request packet to reach the server and for its response to get back to the client, thereby completing the request. Logically, the minimum possible round-trip time T_{\min} can be defined as

$$T_{\min} = t_c + t_q + t_p + t_r + t_n$$

$$t_q = s_q / bw$$

$$t_r = s_r / bw$$

where bw is the bandwidth of the network connection, and s represents packet size, with the q and r subscripts indicating “query” and “response”. The parameter t_c is the one round time delay inherent in TCP connection, $t_q + t_r$ is the time it takes to send the request (query) and receive the reply, and t_p is the time spent at the server. Any network latency due to WAN/LAN connections, routers, modems, and so forth is represented by t_n .

4. **Error rate** or errors-per-second is the measure of the number of HTTP requests lost or not handled by a server. This event could appear as a “connection refused” error when attempting to create a TCP/IP connection, as a time-out on creating the connection, or as corrupted data arriving at the client. A lower number indicates better performance. The user’s perception of reliability depends on a low error rate.

Most reported measurements of server performance focus on *throughput* and *connections established per second*. While each of the four metrics described can be relevant and interesting, they are not of primary importance to the user. The fact is that users tend to care most about latency and avoid Web pages that take long to download. Connection,

request and network delays are the three major latencies. Web clients also add delay that is associated with parsing the retrieved data and displaying it for the user. The time required to retrieve and display a series of Web pages is defined as User's Perceived Performance (UPP) [Mogul 1995]; this is the ultimate measure of the success of Web servers.

3.2 Web Traffic Characteristics

The widespread use of the WWW and related applications place substantial performance demands on the network servers. The ability to measure servers' performance is important for tuning and optimizing the servers' architectures and for obtaining an in-depth understanding of the statistical properties of Web traffic can contribute substantially to the development of the required. Many studies have identified important Web traffic characteristics. For example, Arlitt [1996] used six different data sets to identify some invariants in Web traffic (three from academic environments, two from scientific research institution, and one from a commercial Internet provider). These data sets led Arlitt to conclude that HTML and image documents account for 90-100% of total requests to servers, that the mean transfer size is quite small (5-21Kbytes), and that the file size distribution is Pareto (see Table 3.1 for details on the Pareto, Lognormal and Inverse Gaussian distributions, each of which is highly skewed and heavy-tailed¹). In Arlitt's study, 10% of accessed files accounted for 80-95% of server requests and 90% of

¹ A heavy-tailed distribution has the property the upper tail declines slowly, for example according to a power law, i.e.,

$$P[X > x] \sim x^{-a} \quad 0 < a \leq 2$$

where $a(x) \sim b(x)$ means $\lim_{x \rightarrow \infty} a(x)/b(x) = c$ for some constant c . Random variables whose distributions are heavy tailed exhibit very high variability; in fact, their variance is indefinite, and if $a \leq 1$, their mean is also infinite.

The simplest heavy-tailed distribution is the Pareto distribution. The Pareto distribution is hyperbolic over its entire range; its probability mass function is

$$p(x) = ak^a x^{-a-1}, \quad a, k > 0 \quad x \geq k.$$

and its cumulative distribution function is given by

$$F(x) = P[X \leq x] = 1 - (k/x)^a$$

The parameter k represents the smallest possible value of the random variable.

bytes transferred and that there is high degree of variability in the size of files transferred. The analysis of the extended logs of a 1994 Californian congressional elections Web server [Mogul 1995] showed no correlation between file size and connection time for files under 30kBytes. In this analysis, the majority of traffic was generated by transfers of small images and request arrivals did not appear to follow a pure Poisson process. This result means that the requests may not follow a renewal process, thus making it more difficult to simulate arrivals and to parameterize the traffic intensity. Work on probabilistic models [Huberman 1998] [Pirolli 1999] of the number of pages that a user visits within a Web site showed that an inverse Gaussian distribution is appropriate for this parameter. Statistical characteristics of the Web request traffic patterns in dynamic and heavily accessed Web server environments have been analyzed in several studies [Squillante 1999] [Squillante 1999] [Iyengar 1999]. Some of the most important distributional models of the Web traffic are summarized in next section.

Model	Probability Density Function	Graphs of pdf's
Lognormal	$p(x) = \frac{1}{xs\sqrt{2\pi}} e^{-(\ln x - m)^2 / 2s^2}$	
Pareto	$p(x) = ak^a x^{-(a+1)}$	
Inverse Gaussian	$p(x) = \sqrt{1/2x^3} p e^{-1(x-m)^2/2xm^2}$	

Table 3.1 Probability density functions (pdf's) of three models

3.2.1 Distributional Characterization of the Web Traffic

Before we proceed with the distributional characteristics of the Web traffic, we need to define the components of the Web traffic. Web files usually include other files by reference, such as images and thus the user's request for a single Web page often results in the transfer of multiple files. The set of files in a Web page is called a *Web object*. In order to retrieve a Web object as rapidly as possible, the HTTP protocol allows several simultaneous connections to retrieve sets of files. This method is faster than accessing individual files at well-spaced intervals. Eight distributional models of the Web traffic are

now introduced. Typical values for the various parameters are given in Table 3.2 at the end of this section.

- 1) Crovella [1995] [1997] demonstrated and explained that Web traffic can be self-similar. This characteristic is quite different from traditional traffic models that use Poisson or Markovian modeling and which predict that longer-term² correlations should rapidly die out. Under standard traffic models, traffic that is observed on large time scales should appear quite smooth [Floyd 2001]. Self-similarity in traffic has been shown to have a significant negative effect on network performance [Erramilli 1996] [Park 1996]. The **high variability** [Arlitt 1996] [Crovella 1997] and **self-similar nature** of Web access load is modeled through heavy tail distributions such as Pareto, Lognormal and Weibull functions. Random variables generated by these distributions can assume extremely large values with non-negligible probability.
- 2) **File sizes.** Web workload is influenced by the distribution of file size on the server. In the case of heavy-tailed distribution of file sizes, the server's file system has to deal with highly variable file sizes [Bray 1996] [Crovella 1997] [Arlitt 1996]. This feature determines that the retrieval time for the files is also highly variable. The upper tails of the distribution (the distribution of extreme values) for file size are well described by the Pareto distribution with shape parameter $\alpha < 2$ [Floyd 2001]. For $\alpha < 2$, the Pareto distribution has infinite variance. Crovella [1997] showed that the effect of adding multimedia files to a set of text files increases the weight of the tail. The body of file size distribution tends to follow the Lognormal function [Barford 1999] [Arlitt 2000] [Pitkow 1999].
- 3) **Request Sizes.** The distribution of request sizes can be quite different from the distribution of file sizes because requests can result in any individual file being

² "Longer-term" here means time scales from hundreds of milliseconds to tens of minutes.

transferred multiple times, or not at all. File sizes refer to those files stored in the file system of the server and request sizes correspond to the files transferred over the network from the server. Accurate modeling of the request sizes is important for accurately exercising the server and the network. Empirical measurements show that the distributions of request sizes can show heavy tails [Arlitt 1996]. Web pages with a size larger than 50kB make up 10 percent of the total number of pages but account for 80 percent of the traffic volume. Most requests are made for small groups of files in the server.

- 4) **Popularity.** The relative number of requests made to individual files on the server is the fourth property of the workload. Popularity measures the distribution of requests on a per-file basis. Even when the distributions of file sizes and request sizes are fixed, there still can be considerable flexibility in how requests are distributed among individual files. Popular files typically tend to remain in caches, so the distribution of popularity greatly influences the behavior of caches.

Popularity distribution for files on Web servers has been shown to commonly follow Zipf's law [Glassman 1994] [Cunha 1995] [Almeida 1996] [Huberman 1998]. Zipf's Law states that if files are ordered from most popular to least popular, then the number of references to a file (P) tends to be inversely proportional to its rank (r). That is : $P = kr^{-a}$ for some positive constant k . The empirical measurements of the exponent a are often quite close to -1, which expresses Zipf's Law in its original form. The heavy tailed nature of the Zipf distribution means that only a few files account for the majority of requests.

- 5) **Embedded References.** HTML pages usually are often comprised of other files such as icons and pictures. In order to capture the structure of these files, it is important to characterize the distribution of embedded references in a HTML page. The Pareto distribution has been shown to give the best fit [Barford 1998] [Pitkow 1999].

- 6) **Temporal Locality.** Another characteristic of the Internet that plays an important role on the efficiency of caching algorithms is the temporal locality of HTTP requests.

Temporal locality in Web workloads refers to the likelihood that the same document is re-referenced within short intervals. A usual way to measure temporal locality is to compute the “stack distance” or distances between requests to the same document.

This measurement is achieved by counting the number of different requests between requests to the same documents. One way to measure temporal locality is using the standard LRU (Least Recently Used) stack-depth analysis [Arlitt 1996]. The stack distance sequence can be generated as follows for any given sequence of requests. A file is initially requested and automatically placed on top of the stack, pushing other files down. On subsequent requests, its current location is recorded, and the file is moved back to the top of the stack. The depth at which each requested file is found is the request’s stack distance.

Stack distance captures temporal locality. A small stack distance corresponds to file requests occurring close to each other in the reference stream. Assuming the initial order of the stack is known, the stack distance sequence contains information equivalent to the request sequence and each can be obtained from the other. The distribution of stack distances for any Web request sequence can thus serve as a measure of the temporal locality in the sequence.

It was observed in [Almeida 1996], [Pitkow 1999] and [Tauscher 1997] that typical distribution for request sequences is Lognormal. As the Lognormal distribution has most of its mass concentrated in small values, this result indicates that significant temporal locality is often present in Web request sequences.

- 7) **OFF Times.** Proper characterization of inactive OFF times (when a user is thinking) is necessary to replicate an accurate transfer of Web objects and capture the bursty nature of Web user’s requests. The think time parameter governs the amount of time clients wait between the response from the server and the next request to the server.

This parameter is typically modeled using a Pareto distribution [Barford 1999] [Pitkow 1999] [Deng 1996]. “The heavy-tailed distribution of user think times also seems to be a feature of human information processing.” [Crovello 1995]

- 8) **Session Lengths.** The session length refers to the number of page requests a client makes before it closes a maintained connection. A user normally surfs through several Web pages within a site before proceeding to another site. The session length has been shown to approximately follow an inverse Gaussian distribution [Pitkow 1999] [Catledge 1995] [Cunha 1995] [Huberman 1998]. For the HTTP/1.1 protocol, this parameter strongly affects the time interval that a persistent connection stays connected.

Category		Distributions	Parameters
File Size	Base (30%)	lognormal (93%)	$m = 7.630$; $s = 1.001$
		Pareto (7%)	$a = 1.0$; k=10000; min=75
	Embed (38%)	lognormal	$m = 8.215$; $s = 1.460$
	Loner (32%)	lognormal (66%)	$m = 7.101$; $s = 1.200$
		lognormal (34%)	$m = 11.151$; $s = 1.143$
Request Sizes		lognormal (93%)	$m = 7.881$; $s = 1.339$
		Pareto (7%)	$a = 1.177$; k=34102
Popularity		Zipf	
Embedded References		Pareto	$a = 1.245$; k=2; max=150
Temporal Locality		lognormal	$\ln(m) = 4.214$; $\ln(s) = 2.187$
OFF Time		Pareto	$a = 1.4$; k=1; max=1800
Session Length		Inverse Gaussian	$m = 3.86$, $l = 6.08$; max=95

Table 3.2 Distributions and parameters of several Web characteristics used in this project

3.3 Workload Generator of Web Traffic

Some researchers have tried to directly evaluate the performance of Web servers and proxies using real workloads. These measurements are important for understanding

server performance, for diagnosing server problems and also for comparisons with synthetically generated loads. Most recent studies on Web server performance concentrate on creating workload generators in a test-bed environment [SPEC 2001] [Mindcraft 2002] [Mosberger 1998] [Cluts 1998] [Barford 1998]. Trace-driven simulation and analytic simulation are two approaches to workload generator design. Trace-driven workload generation uses prerecorded traces and either samples or replays the records to generate workloads [Barford 1998]. Analytic workload generation uses statistical models based on various workload characteristics to generate traffic that attempts to mimic the behavior observed in actual networks.

Both approaches have strengths and weaknesses. The trace-driven simulation is easy to implement and matches the activity of a known system. Since this approach treats the workload as a “black box”, insight into the causes of the system behavior is difficult to obtain: the trace will always reflect the properties of the system it is measured from.

Trace-driven simulation offers very limited control over the trace, which decreases one’s ability to adjust the workload to investigate controlled variations in traffic. Increase in the Web load may occur along different dimensions that depend on the number of users accessing the server, the files accessed, and the size and complexity of the different pages [Bryhni 2000]. Therefore, it is not straightforward how to extend a trace to represent a higher load level. Some people [Bryhni 2000] aggregate the load by folding a longer time of consecutive Web accesses into a much shorter time trace. However, we do not know exactly how this reflects real access patterns. Analytic workloads do not have these limitations, but they can be challenging to construct and validate. In this approach, it is vital to identify the characteristics of the workloads and to measure them empirically. To create a single output workload accurately exhibiting a large number of different characteristics is difficult to achieve. However, measurements of server performance using synthetically generated loads are useful for demonstrating server performance over

a range of loads and in a controlled fashion. This demonstration would not be possible in real use studies.

Typically, load generating schemes that equate client load with the number of client processes in the test system are used [Cluts 1998] [Banga 1999] [SPEC 2001] [Barford 1998]. Adding client processes is thought to increase the total client request rate. To decrease costs and increase control, it is desirable to use a small number of client machines to simulate a large client population. A set of N Web client processes are executed on P client machines, thus taking advantage of the multithreading feature of the system. In this procedure, the client machines and the server usually share a LAN. Each client process establishes an HTTP connection, sends an HTTP request, receives the response, waits for a certain time (think time), and then repeats the cycle. To make the simulation more realistic, the sequence of URLs requested can come from a database designed to reflect realistic URL request distributions observed on the Web. As for think times, they are determined by making the average URL request rate equal to a specified number of requests per second. N is typically chosen to be as large as possible within the performance capacity of the client machine, thus allowing a high maximum request rate. A typical workload generator is illustrated in Figure 3.1.

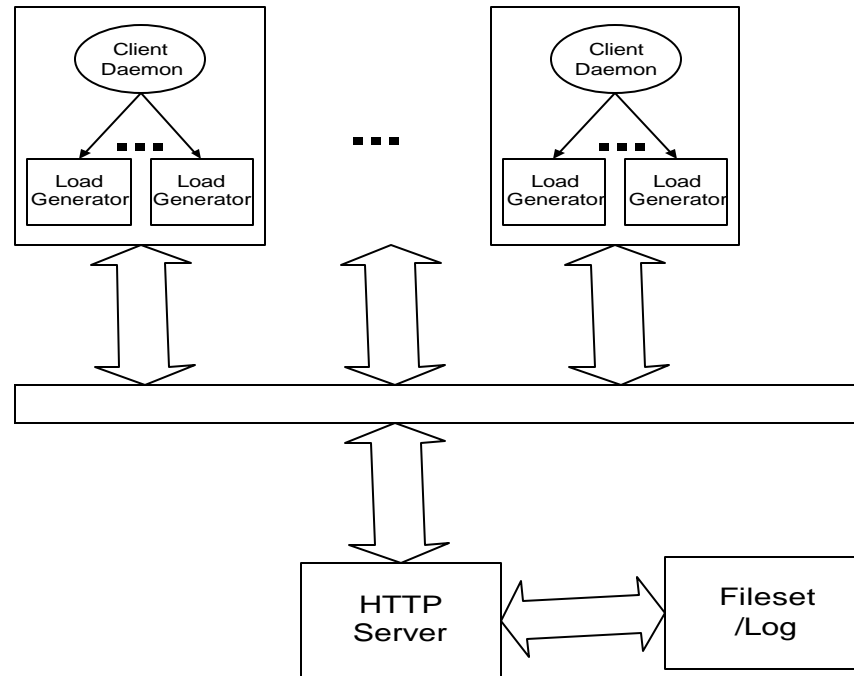


Figure 3.1 Typical workload generator

Instead of pre-determining the think time, as done in a typical workload generator, such as SPECweb[SPEC 2001] introduced in the next section, we will vary the think time by following a statistical model. As mentioned before, the think-time model is important in replicating an accurate transfer of Web objects and capturing the bursty nature of Web user's requests. We introduce some commonly used Web benchmarks in the next section.

3.4 Web Benchmarks

In this section, we introduce five major Web benchmarks: **WebStone** [Mindcraft 2002], **SPECweb** [SPEC 2001], **Httpperf** [Mosberger 1998], **WCAT** [Cluts 1998] and **SURGE** [Barford 1998]. WebStone, SPECweb, Httpperf and WCAT are categorized as simple request generators. These benchmarks are essentially based on the WebStone model, which make repeated requests for some set of files either at a very uniform rate, or as fast as the client system(s) can absorb.

Simple Request Generators

WebStone and SPECweb are the two most accepted benchmarks [Barford 1998] [Banga 1999]. WebStone is a distributed, multi-process benchmark that simulates an arbitrary number of clients randomly downloading pages from the server for a configurable amount of time. As most users access the main page of a Web site and proceed to follow links thereafter, the random access patterns of this simulator are not an accurate representation of behaviors found on the WWW. It initiates a fixed number of processes that constantly download one file at a time as often as possible and measures the throughput and latency of each HTTP transfer. This simulation does not account for periodic idle time by WWW users who surf in and out of sessions. Furthermore, it does not simulate the common practice of WWW browsers, which generate multiple connections to download pages and inlined images concurrently. Servers are represented in the WebStone test by four different file sets. The first set is composed of small pages of less than 20kB, in the second set, file sizes vary between 1 and 100kB, while the third mix is comprised of larger files ranging from 20kB to 1MB. The fourth set contains files varying in size from 0kB to 1MB. These four file sets seem too limited to reflect the files in a typical Web server.

SPECweb is a benchmark developed by Standard Performance Evaluation Corporation (SPEC). SPECweb differs from WebStone in its workload division into a mixture of four file classes by file sizes: less than 1kB, 1 to 10kB, 10 to 100kB and 100kB to 1MB. The weight of each class is obtained from the analysis of existing Web server logs and determines the site's page set, the collection of user profiles and the inter-arrival time between users utilized in simulations. The workload is generated according to the empirical distributions of user inter-arrival times. SPECweb99 directs 35 percent of its activity to the smallest class, 50% to the 1-to-10-kB class, 14% to the 10-to-100-kB class and 1% to the largest files. Access within each class follows Zipf's distribution. Httpperf was developed by HP Research Labs. Httpperf measures the request throughput of a Web

server. For example, HTTP calls and sessions can be generated deterministically at a fixed rate. The sessions consist of several HTTP call bursts spaced by fixed user think times. Repetitive URL requests and URL request sessions can also be generated at a given rate from a fixed set of URLs. The workload generated by this simulator is far from mimicking the real-world traffic patterns observed by Web servers.

The Microsoft Web Capacity Analysis Tool (WCAT) is another tool for testing the client-server configurations. WCAT is designed specifically for evaluating Internet servers running Microsoft Windows NT® Server and Microsoft Internet Information Server (IIS) and provides more than 40 ready-to-run workload simulations. Basic tests include repeated requests for a 1kB file, requests for 12 files ranging in size from 256 bytes to 256kB, requests for 14 files between 256kB to 1MB with total workload of 50MB or 200MB and a WebStone 1.1 sample workload. Similar to the two benchmark programs mentioned above, the WCAT test requires a variety of parameters to be set. The number of client browsers needs to be specified for each test as well as the size, type, and rate of client requests. “ThinkTime”, frequency of requests, pages requested and duration of the test are also required. WCAT also offers the ability to test a server’s performance using ASP, Internet Server Application Programming Interface (ISAPI) extensions and Common Gateway Interface (CGI) applications. However, WCAT shares the drawbacks found in WebStone and SPECweb as discussed below.

In the WWW, HTTP requests are generated by a huge number of clients, each with a think time that follows a distribution having a large mean and variance. Furthermore, the think time of clients is not independent. Indeed, factors such as human users’ sleep/wake patterns and scheduled publications such as news cause high correlation of client HTTP requests. As a result, bursts can be observed in the server’s HTTP request traffic at several scales of observation, and with peak rates exceeding the average rate by factors of 8 to 10. Those peak request rates can easily exceed the capacity of the server.

By contrast, simple request generation methods, such as SPECweb, WebStone and WCAT, have fixed think times or independent think time distributions with small means and variances. In these methods, the goal is to generate a fixed number of requests per time interval. As the number of requests per second increases, the time between consecutive requests decreases. This implies that the clients stay essentially in lock-step with the server. As a result, bursts are rarely found in the traffic.

In next section, an analytic approach is introduced to combat the problems associated with the simple request generation method.

Scalable URL Reference Generator (SURGE)

SURGE is a benchmark tool designed by Paul Barford [Barford 1998] and we have selected it as the basis for the server-tests described in Chapter 4. The source code is publically available and the programs can be compiled on both UNIX and Linux systems. SURGE attempts to imitate a stream of HTTP requests originating from a fixed population of web users. Each user is modeled by an ON/OFF random process (User Equivalent, or UE) that makes requests for Web files during the ON period and lies idle when OFF. A **User Equivalent** is a workload roughly corresponding to that generated by a normal user. The intensity of service demand generated by SURGE can thus be measured in Unit of UE's. In the software, a UE is implemented as an independent threaded process operating in an endless loop alternating between requesting Web files and idling. To be accurate, both the Web file requests and the idle times must exhibit the distributional and correlational properties that are characteristic of real Web users. Within ON periods, users keep the connection alive for multiple retrieves. OFF periods correspond to the user think time. SURGE is based on a set of eight distinct statistical models (see Section 3.2.1) of Web use behaviours in an attempt to capture properties observed in real Web workloads. The file-size model consists of three categories: base files, embedded files and single files. The base files refer to HTML files that contain

embedded files. In SURGE programs, the base files account for 30% of the total pages and follow a hybrid of Lognormal and Pareto distribution. Embedded files refer to files which are referenced by base files, typically inlined images, and account for 38% of the total pages. The file size follows the Lognormal distribution. Single files are files which are neither base nor embedded. 32% of the total pages belong to this category and the sizes follow a hybrid of Lognormal and Lognormal distributions. Table 3.2 lists the distributions and parameters of several Web characteristics associated with the models discussed in Section 3.2.1.

On the WWW, a single file request often results in the transfer of multiple files from the Web server. When SURGE generates an object, one file from the base file set and a number of files from the embedded file set are selected. In this generation, each of the base files are assigned to one and only one object, while embedded files can be assigned to multiple objects. The number of embedded files in each object is determined by the Pareto distribution.

Since each UE has significant idle periods, it is a very bursty process and the resulting traffic is quite different from approaches typically used by other Web workload generators. Barford [1998] showed that SURGE more realistically benchmarks server performance than other commercially available tools. For example, empirical studies of operating Web server have shown that they experience highly variable demands in the form of CPU loads and number of open connections [Mogul 1995] [Barford 2001]; the workload generated by SURGE matches this by causing much higher variability in number of open connections than does SPECweb. In addition, SURGE generates network traffic that is self-similar and bursty at high loads, as is seen in traffic measurements. This capability is not available in other available load generators.

SURGE supports HTTP 0.9/1.0/1.1 protocols. (HTTP 0.9/1.0 uses a separate TCP connection to be used for multiple file, while HTTP 1.1 allows a single TCP connection to be used for multiple files.)

Compared to other benchmarks, SURGE generates much more realistic traffic. But, by using the analytical approach, SURGE also offers other advantages. The models used in SURGE are explicit and can be examined directly by the user. This feature enables a user to artificially inflate different aspects of a site's traffic, permitting the webmaster or researcher to study expected future demands and other alternative conditions. The stress on the server and network is more intense with SURGE than in other Web benchmarks [Barford 1998]. As the test is more realistic and representative of the WWW, SURGE was used by the HTTP-NG³ [W3C 2001] as the basis for its representative testbed because of its analytical strength [Pitkow 1999].

“From these empirically derived distributions, SURGE creates a mock Web site that essentially contains page with embedded images that match the statistical properties measured from real Web site.” [Pitkow 1999]

The basic configuration parameters of the models, described in Section 3.2, have not been changed from those used by [Barford 1999] in his performance evaluation of HTTP protocols. They were developed on the analysis of empirically measured Web workload traces [Barford 2001].

In this study, all of those parameters are fixed except for s of the Lognormal distribution for the “loner” file size. This parameter is intended to represent the larger, streaming audio and video files found on multimedia servers. Usually, an audio or a video file only appears as a link in a file page. In other words, instead of being downloaded simultaneously with other pages, those files have to be downloaded separately. These files are also large compared to regular image files. We want to see the effect of average file size on the performance of a cluster.

³ “Between July '97 and Dec '98, the HTTP-NG Activity explored the future development of the HTTP protocol. The motivation was the impression that HTTP/1.1 is becoming strained modularity wise as well as performance wise. The HTTP-NG Activity produced a number of proposals that successfully addressed these issues, which were presented to W3C members and at an IETF meeting in Dec. 98.” [W3C 2001] W3C (2001). Final HTTP-NG Activity Statement.

3.5 Summary

In this chapter, several distributional characteristics of Web traffic were discussed. We then introduced five commonly used Web benchmarks: SPECweb, WebStone, Httpperf, WCAT and SURGE. SURGE was chosen as our server stress-testing tool. In next chapter, we will show how we used SURGE to stress-test the server and developed server models from the results.

Chapter 4

Server Stress Tests and Server Models

This chapter describes the server stress testing experiments. Server processing models are developed on the results from these experiments. We then integrate these models into the simulation system that we use to study server clustering techniques and validate them.

4.1 Experimental Setup

The environment for the experiments presented in this document consisted of one HP Kayak XU PC running RedHat Linux 7.1(Linux 2.2.16), as well as one Sun Untra-2 and two Sun Ultra-10 workstations. The Untra-2 possesses 128MB of RAM with a 300MHz UltraSPARC processor. Each Untra-10 has a 440MHz UltraSPARC processor. One of them is provided with 128MB of RAM and the other with 64MB RAM. The PC is configured with dual Pentium II 333 CPU, 256MB of RAM and SCSI 10000 RPM Seagate “cheetah” disk drive. The server runs Apache 1.3.12. The configuration for the Apache software was taken from [Apache 2002]. The workstation specifications are summarized in Table 4.1.

Server Hardware	Dual Pentium II 333, 256 MBytes RAM, 512KB cache, SCSI Seagate 10000RPM HD			
HTTP Server Software	RedHat Linux 7.1; Apache 1.3.14			
HTTP Client Hardware	jackanapes	SUN Ultra-10: 440MHz UltraSPARC-IIi processor with 2-MB external cache, DRAM with speed of 50ns	Memory	128 MB
	jacobi			64 MB
	jabberwock	SUN Ultra-2 with 300MHz CPU and 128MB memory		
HTTP Client Software	Sun OS Release 5.8, CDE1.4, X11 Version 6.4.1			

Table 4.1 Experiment specifications

Due to the limitations of the research resource, a stand alone local area network is impossible in this research; all tests were run after midnight or over the weekend to decrease the amount of unrelated traffic to a minimum and to make the results repeatable. Each run was repeated 3 times in order to make up for network fluctuations and averages were then taken from these results. Figure 4.1 demonstrates that the server, “jawbreaker”, is connected through a 10Base_T hub. The three other UNIX machines are connected separately through a 10 Mbps hub. The four hubs are joined together by a 1Gbps backbone. Average roundtrip time as measure by “traceroute” command was 1.581ms (variance = 0.094). Bandwidth as measured by repeated FTP requests of a 3.18MB file was 0.842Mb/s (variance = 0.227Mb/s). These measurements took into consideration the 3 hops between the clients and the Web server.

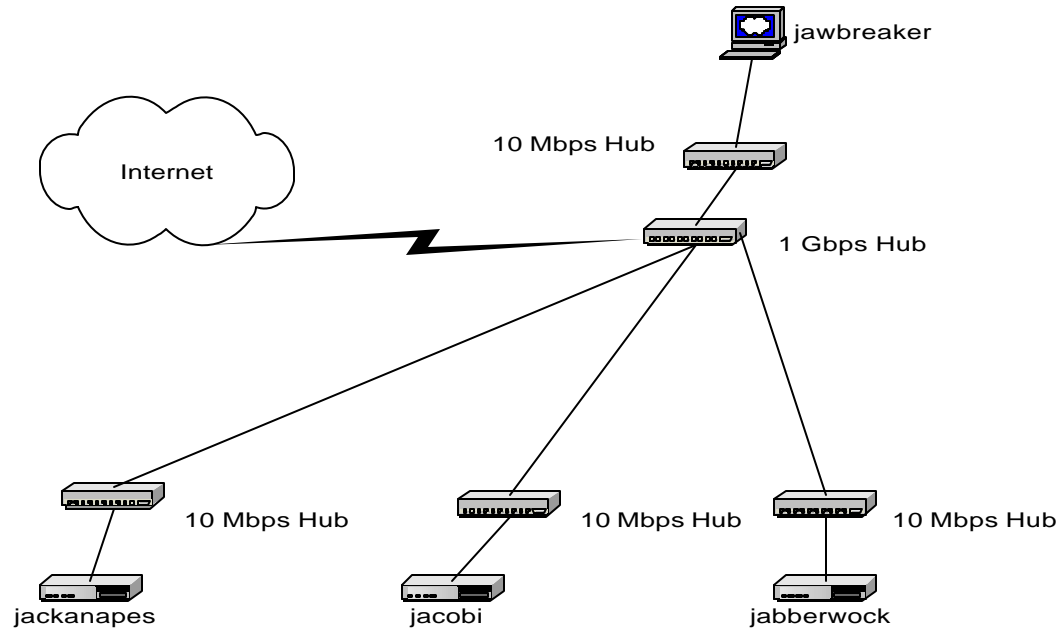


Figure 4.1 Experiment setup

Another popular server, the IIS 4.0 was also tested on a Dell server station running Microsoft NT 4.0 with 350 MHz CPU and 256 M 100MHz ECC SDRAM (100MHz system bus speed, Ultra2/Wide SCSI, 10,000RPM hard disk). Unfortunately, however, this server became very unstable when the load almost reached 320 UE's. Adding more UE's exhausted the resources in the operating system (number of processes, available memory, etc.) and crashed the machine. The Linux server was still able to run smoothly when put through the same test. Our study therefore concentrated on the Apache server (with a Linux operating system).

Two key parameters control the number of connections to the Apache server: the maximum number of connections (MaxClients) and the timeout beyond which an idle connection is cut off (KeepAliveTimeout). The maximum number of connections largely depends on the memory and CPU capacity of the web server. The KeepAliveTimeout parameter, however, is set in accordance with the arrival traffic pattern. Two sets of possible configuration parameters are given in Table 4.2. They are defined in httpd.conf

file, which is used to configure the operations of Apache server. The configuration for Experiment I was based on the `highperformance.conf` file supplied with the Apache distribution and was used in Barford's experiments [Barford 1999]. In Experiment II, the server was configured differently in an attempt to take in consideration the specifications of the computer used in this study. In Section 4.3, you will see a comparison of the results under these two configurations. Minor optimizations could have been made to the web server such as turning off logging or adding more memory. Such optimizations might have improved the server's performance slightly, however, in this study we attempted to use a consistent set of test conditions in order to accurately compare the results from different experiments and obtain good performance. The goal was not necessarily to achieve the highest throughput numbers possible.

Parameter	Experiment I	Experiment II
KeepAlive	On	On
MaxKeepAliveRequests	150	150
KeepAliveTimeout	15	15
MinSpareServers	5	5
MaxSpareServers	10	20
StartServers	5	8
MaxClients	150	256
MaxRequestsPerChild	Unlimited	Unlimited

Table 4.2 Apache 1.3.12 configuration parameters

4.2 Assumptions and Restrictions

The data reported in this chapter are from a series of experiments on one set of systems and networks; other systems may have totally different performance and care must be taken in extrapolating our results to significantly different configurations/hardware. The numbers reported are intended to reflect the general pattern of Web servers' responses and to illustrate a strategy for investigating the performance of Web servers and load

balancing algorithms. In order to obtain realistic measurements with different hardware the server models would need to be recalibrated with a new set of SURGE measurements.

Two limitations of our study are as follows:

- **Local Area Network Latency:** Tests were run in a local area network, providing shorter delays than a wide area network would in real life. As a result, file transfer time and connection durations are shorter than those over the Internet.
- **No Packet Loss:** The loss characteristics of WAN's and LAN's are very different. Packet losses caused by network congestion and other factors are much more common in wide area than in local area networks. This study does not evaluate the performance aspects of a server dependent on such network characteristics. In other words, an error-free environment is assumed in this project. It is well known that packet losses and delays increase the lifetime of the server's connections, thus inducing a large number of simultaneous connections that can cause a significant performance degradation of the server. These considerations need to be factored in when interpreting the results.

4.3 Server Stress-test Results

As mentioned in Chapter 2, users care about latency. Measurements of the latency reflect the average time between request of the user and receipt of the Web object by the user.

To insure an accurate representation of server's capacity, we adopted the objects/second parameter as the measurement of the object throughput, instead of using the number of requests per second. Users make requests for objects that may consist of an HTML file along with several embedded references. HTTP/1.1 allows several requests to be made in a persistent connection. In this case, measuring the object throughput is more appropriate and more directly related to the loads (such as, the number of concurrent open connections) on the server. As ISP's can not easily identify objects in server traffic logs, the number of files in an object is normally extracted from the traces by identifying

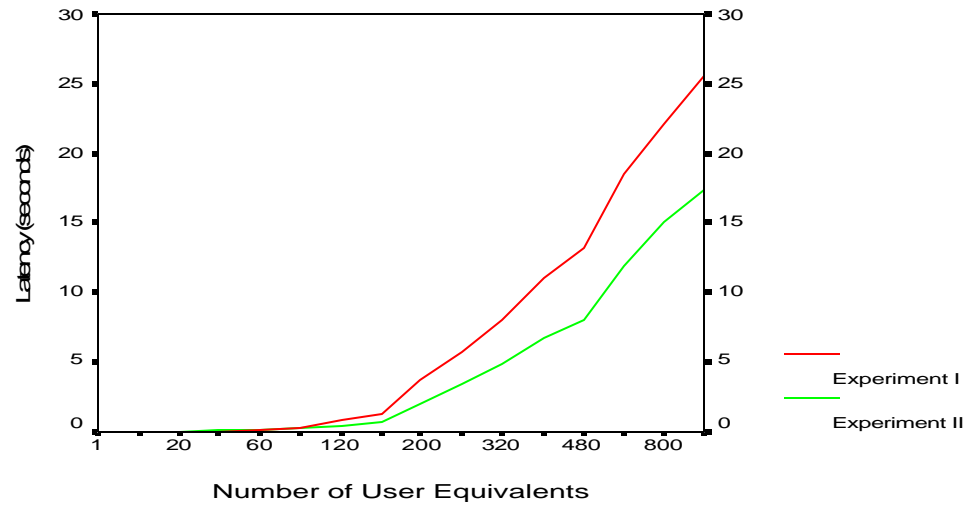
sequences of files fetched by a given user for which the time between transfers is less than the threshold (for example, one second) [Barford 1998]. In our experiments, SURGE is built on the idea of objects and we can easily track down the objects, we decided to take advantage of this opportunity. Both object throughput and byte throughput are measured in this study.

Each experiment in this study exercised the server over a range of loads between 1 and 960 UE's and was performed for ten minutes. An average was taken for three runs of each load (see Appendix B for measurements, average and standard deviation). Each experiment was also started for ten minutes prior to data collection to allow sufficient time to fill up the server's memory with the most requested files, and thus stabilize the effects of subsequent experiments [Barford 1999]. SURGE was configured with 2000 unique files. Varying the parameter s of the loner file resulted in server data-sets with sizes between 52MB and 496MB. This increase of s reflects the expected increasing usage of audio and video contents on the Internet. Table 4.3 lists the total, mean and maximum sizes of the file sizes generated by SURGE.

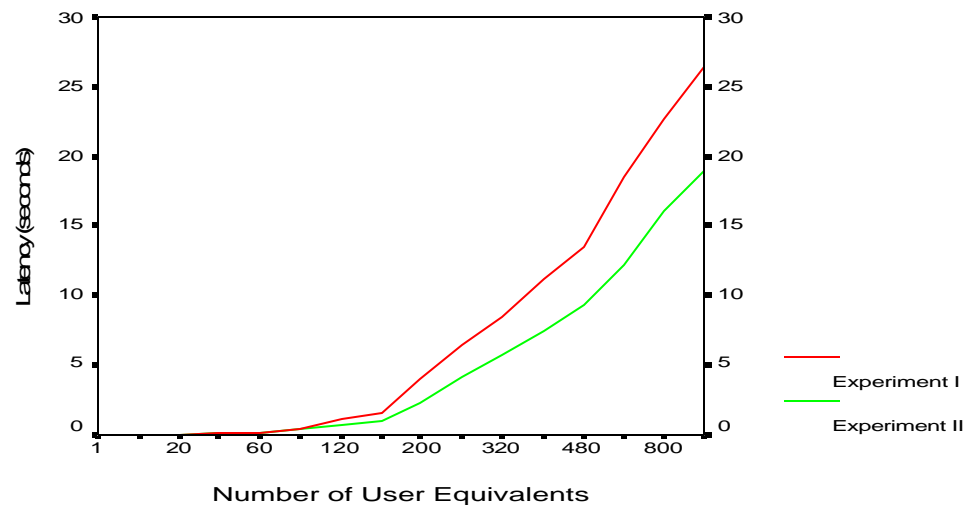
	$s = 1.143$	$s = 1.7145$	$s = 2.5$
Total	51,792,709	103,791,312	496,210,392
Mean	25,896	51,895	248,105
Maximum	3,119,822	10,425,375	103,441,756

Table 4.3 File sizes generated by SURGE

The variability of the three runs for each load is quite small as can be observed from Tables B.3 – B.18. Figures 4.2 - 4.4 illustrate the latency results for the two experimental configurations and the three value of s . It is seen that Experiment II produced far lower average latencies when the load was increased and its configuration parameters were therefore adopted for the remainder of this study.



4.2 Comparison of latencies for two Apache configurations ($s=1.143$)



4.3 Comparison of latencies for two Apache configurations ($s=1.7145$)

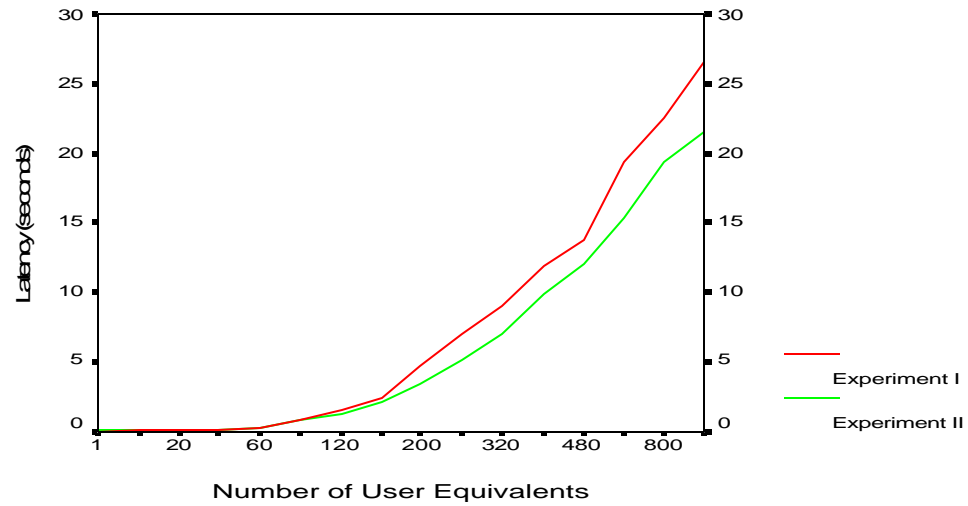


Figure 4.4 Comparison of latencies for two Apache configurations ($s=2.5$)

As shown in Figure 4.5, when the load increased on the server, the time required to serve a Web object increased gradually up to 140 UE's. This result is true for all file size configurations. After 140 UE's, the latency time shoots up sharply. This characteristics is typical for Web servers [Slothouber 1996]. Increasing the average file size also increases the average delay time.

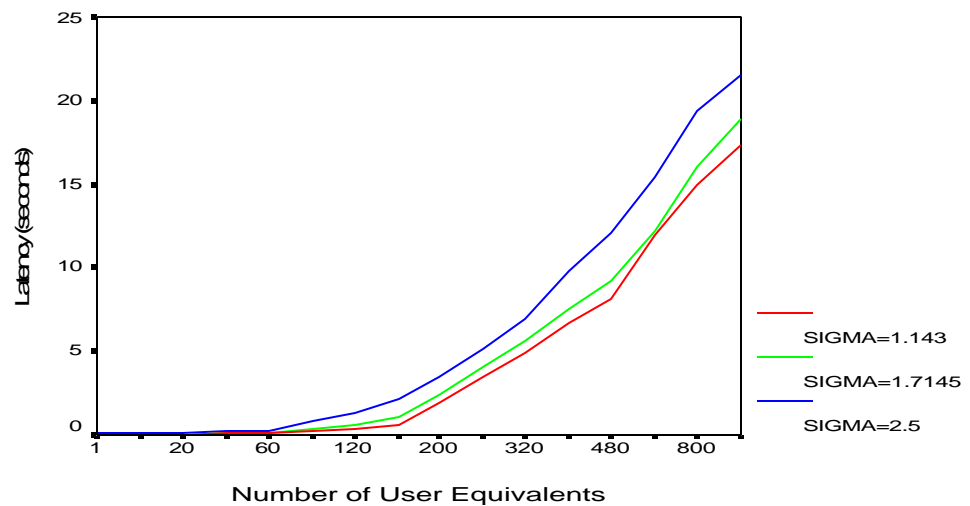


Figure 4.5 Latencies for Apache 1.3.12 (Experiment II)

From Figures 4.6 and 4.7, we can see that a heavy load pushes the server into an overload state. Interestingly, the overload threshold decreased with the increase of the average file size. After 320 UE's, some connection attempts were denied (see Appendix B for the percentage of denied requests) and SURGE's clients gave the error message "trouble connecting to sockets". In SURGE, those clients with dropped connections simply resent them resulting in a deterioration of the overall throughput [Levy-Abegnoli 1999]. This denial of request is associated with high rate of opening and closing TCP/IP connections for long periods of time and is one way that the Apache server uses to protect itself from crashing or increasing the response time suddenly towards infinity. As demonstrated by Slothouber [1996], a server allowing unlimited simultaneous connections creates a deadlock when its resource utilization approaches 100%, since its response time sharply climbs towards infinity. To serve the requests already in the queue and to prevent a dramatic deterioration in the quality of service when the maximum capacity is reached, the server should deny any new file requests. This factor increases the average latency and has a negative effect on the overall throughput of the system (Figures 4.6 and 4.7). The percentages of denied requests (number of timeouts / number of objects served) are illustrated in Figure 4.8. These percentages are integrated into the server model utilized in our simulation and will be discussed further in this chapter.

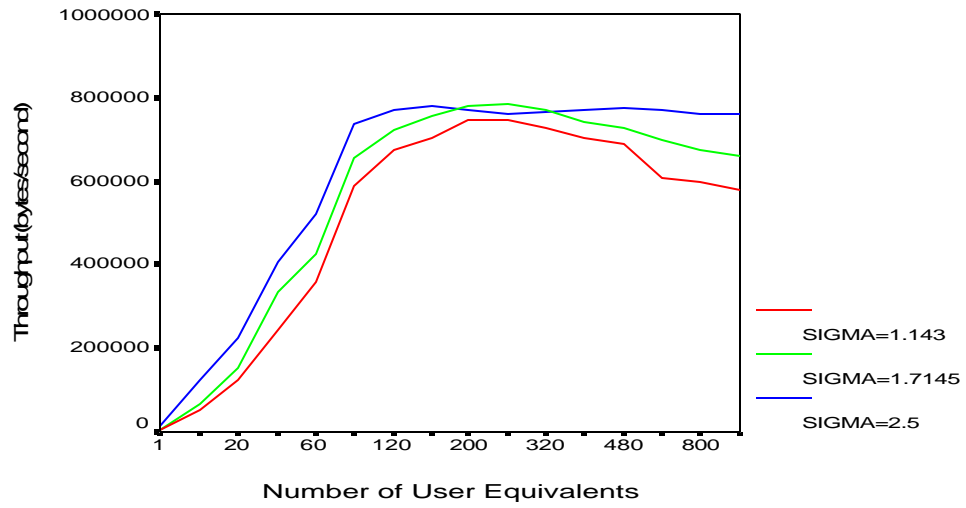


Figure4.6 Data throughputs (bytes/second)

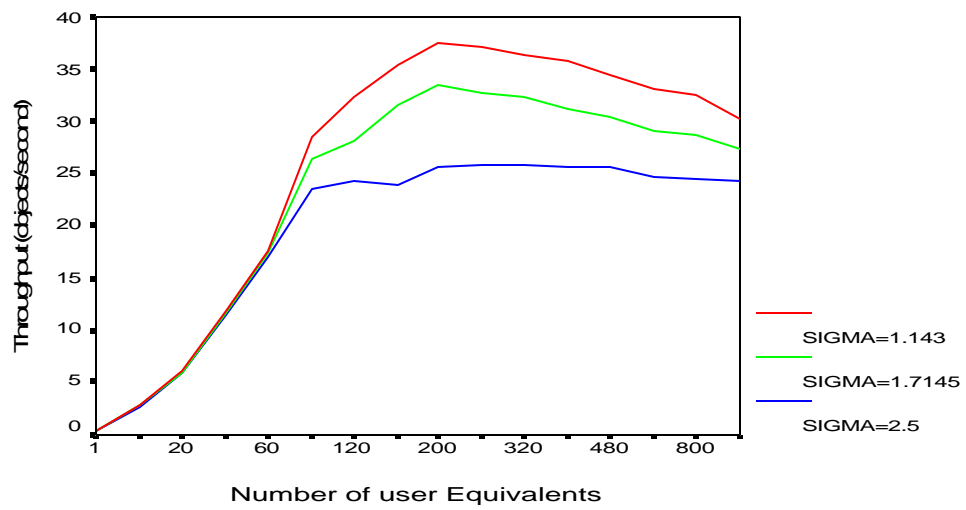


Figure4.7 Objects throughputs (objects/second)

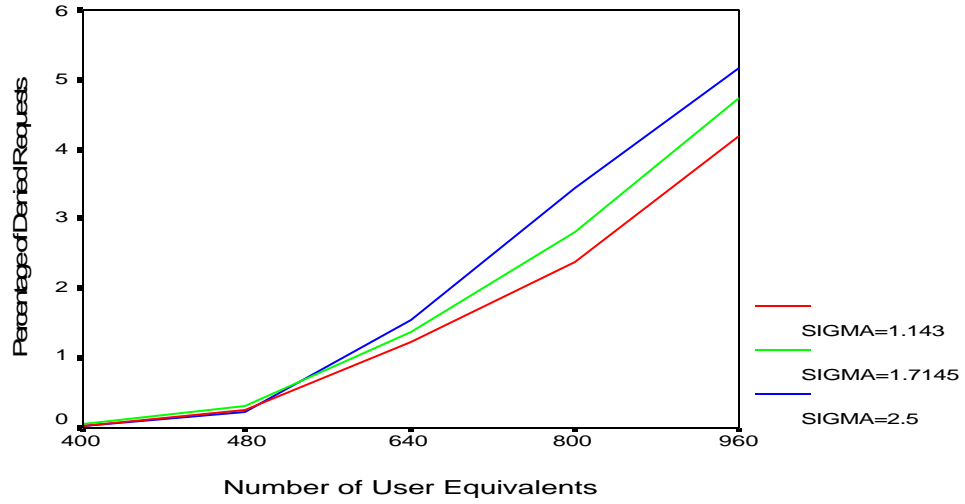


Figure 4.8 Percentage of denied requests

4.4 Development of the Server Models

Our goal is to develop formula-based server processing models that are amendable to integration into an *ns2*-based simulation system. Our approach is to model the server as variable delay, with the amount of the delay a function of the load. As indicated in the figures above, we can experimentally relate the observed delay to the number of UE's generated by the SURGE clients. Users tend to request several objects in a session before moving to another Web site. In our simulation system, when a client makes the first object of a session, the number of UE's N_{UE} increases by one; after the client receives the last object of the session, N_{UE} drops by one. In this section, we apply polynomial fittings (via MATLAB) to the curves in Figure 4.5 in order to parameterize the relationships.

Due to the trend of the three curves in Figure 4.5, it is easier to fit the latency curve in two non-linear regression fittings as demonstrated in Figures 4.10 - 4.12. In fact, one regression fitting was attempted to describe the entire curve with unsatisfying results. Regression fittings need to be simple and accurate, which means keeping the order of the polynomials as small as needed to keep measurements stay on the curve or at least very

close to it. The entire non-linear fitting must show an upward trend in order to be realistic. Therefore, the second fitted curve must start higher than the end of the first one for each individual graph. Some data points were used to generate both fitted curves, helping to create a smoother transition. In Figures 4.9 - 4.11, the top graph shows the overall fitting from 1 UE to 960 UE's. The bottom left one shows the fitting from 1 UE to 140 UE's, the bottom right one shows the fitting from 140 UE's to 960 UE's. All three overall fittings are illustrated in Figure 4.12. Both measured and predicted results are tabulated in Table 4.4. As we can see, each pair of the measurement and predicted value from the curves are quite close which indicates good non-linear fittings. These curves were used in our simulation to determine the server delay for a request as a function of N_{UE} . More details come in Section 4.5. These models do not take into consideration of the specific file size, which could be an area for improvement for future work. A fixed delay time is used for each N_{UE} and s .

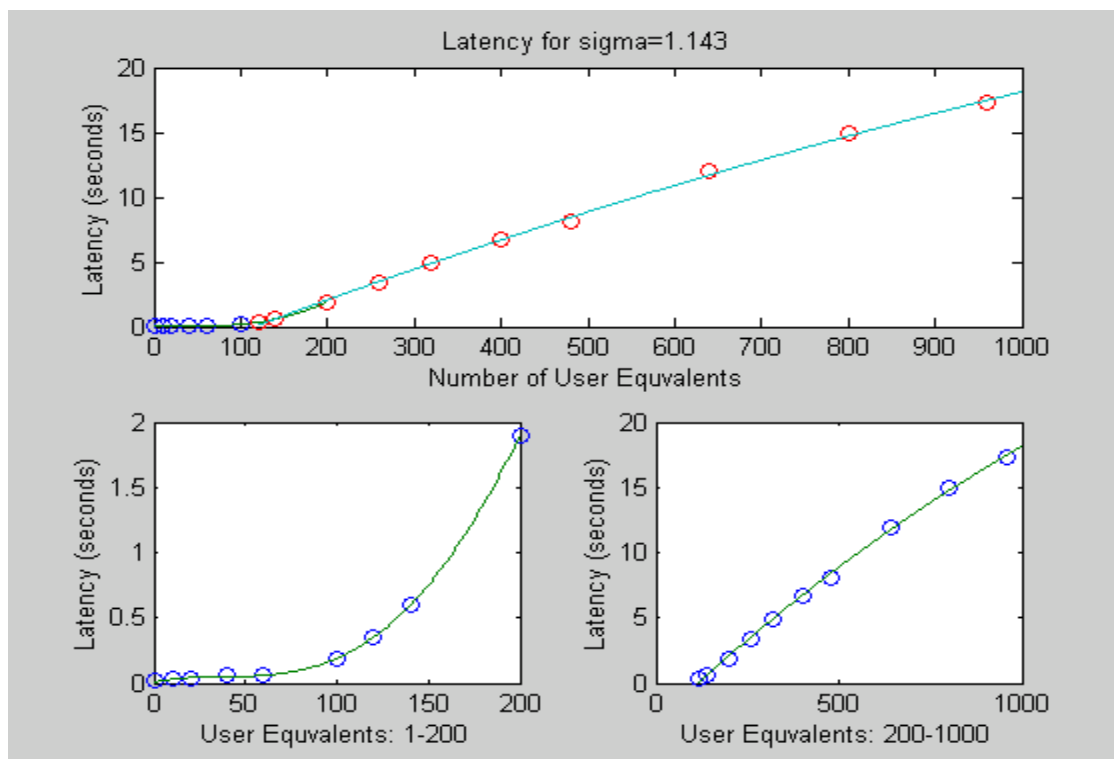
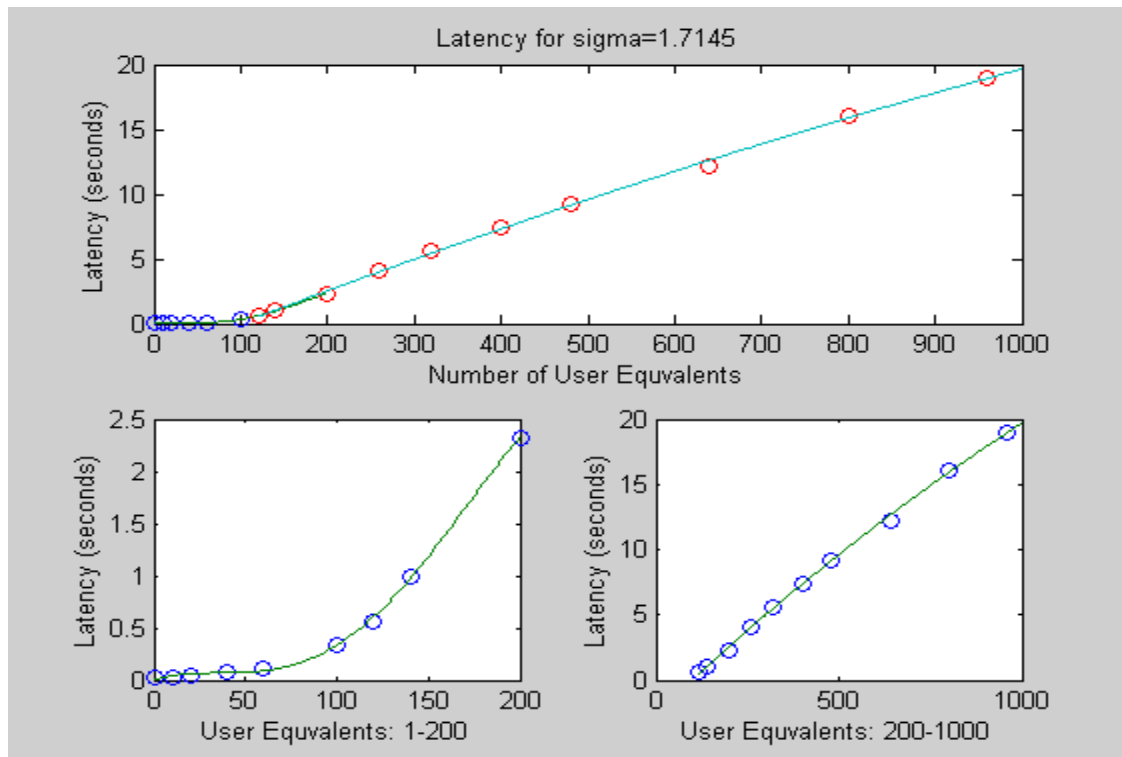
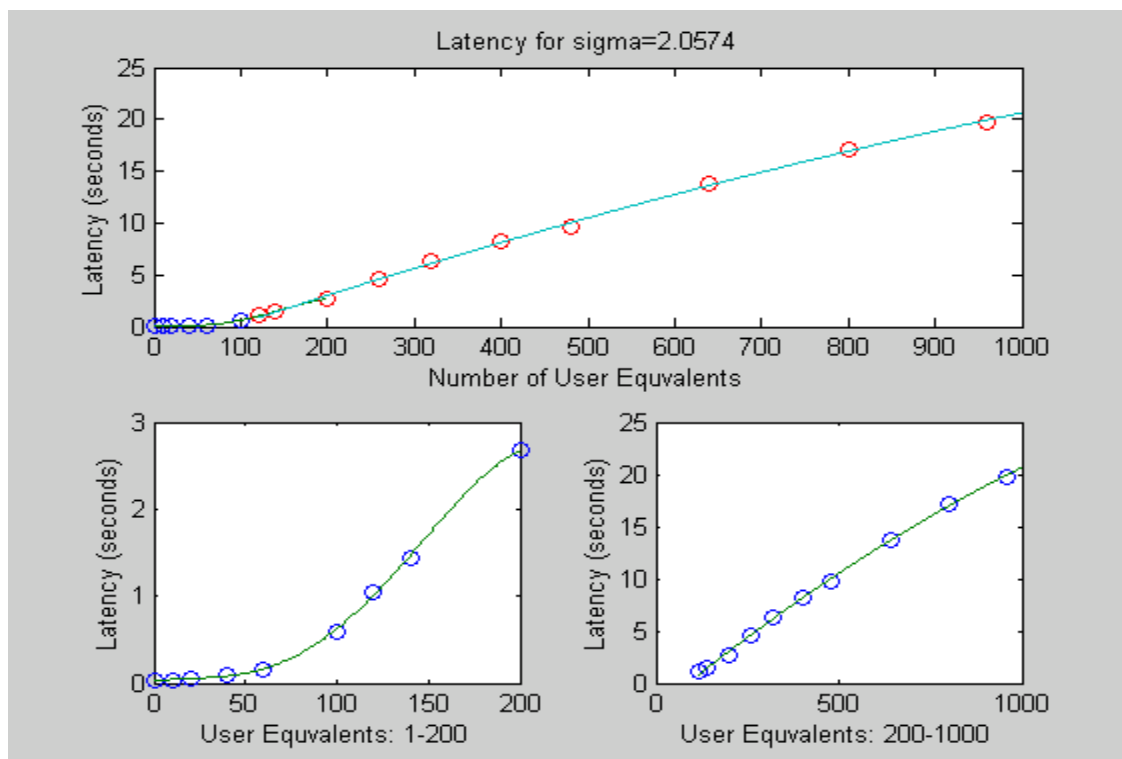
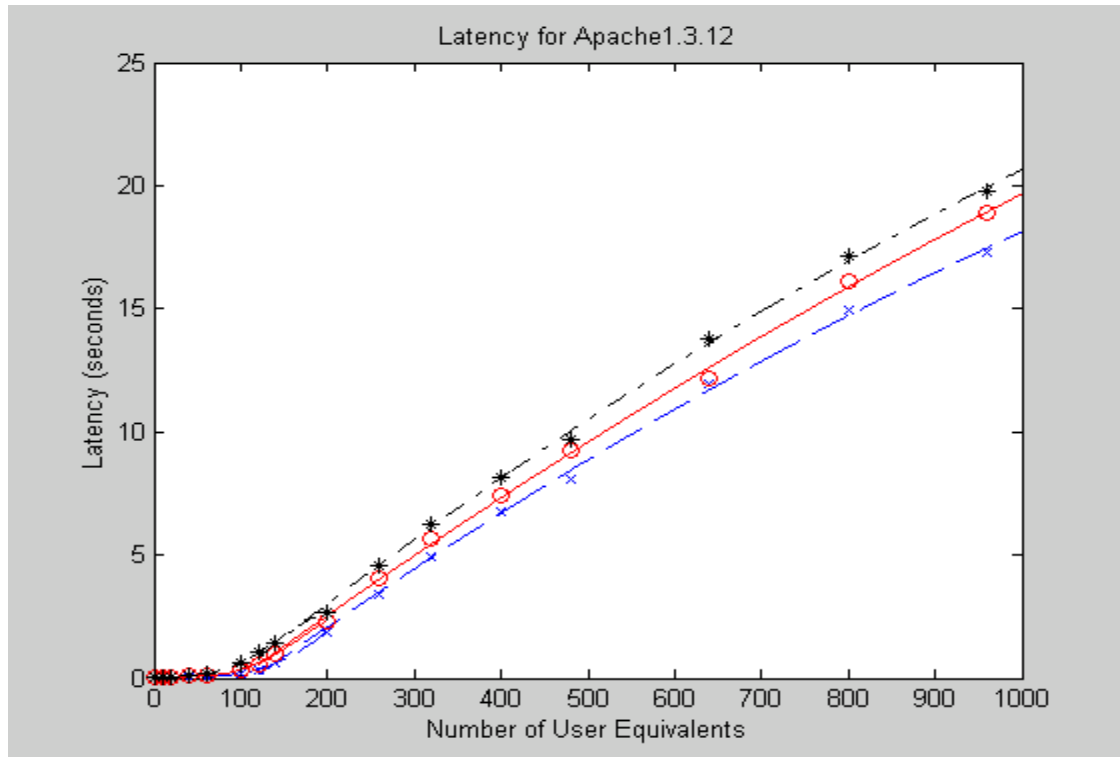


Figure 4.9 Regression fittings for $s = 1.143$

Figure 4.10 Regression fittings for $s = 1.7145$ Figure 4.11 Regression fittings for $s = 2.5$

Figure 4.12 Regression fittings for 3 S 's used

Number of UE's	$S = 1.143$		$S = 1.7145$		$S = 2.5$	
	Measured	Predicted	Measured	Predicted	Measured	Predicted
1	.016	.014	.020	.022	.058	.053
10	.028	.029	.032	.029	.059	.069
20	.034	.039	.038	.041	.075	.080
40	.056	.051	.082	.073	.143	.113
60	.066	.065	.105	.119	.174	.206
100	.183	.190	.344	.329	.796	.771
120	.353	.346	.563	.575	1.306	1.326
140	.598	.600	.998	.995	2.112	2.107
200	1.892	2.131	2.327	2.597	3.402	3.832
260	3.420	3.415	4.050	3.970	5.066	5.443
320	4.907	4.695	5.637	5.331	6.922	7.025
400	6.735	6.396	7.428	7.127	9.789	9.090
480	8.082	8.090	9.213	8.901	12.062	11.104
640	11.957	11.458	12.181	12.383	15.402	14.979
800	14.956	14.799	16.082	15.778	19.350	18.649
960	17.268	18.113	18.912	19.085	21.546	22.114

Table 4.4 Measured and predicted mean latencies of Web objects

4.4.1 Server Models

We present in the following three server processing models and their related models for the percentage of denied requests when the load threshold is crossed. The server processing models are used for calculating the server delay time as a function of the number of user equivalents. The use of models for the percentage of denied requests will be discussed in the next section.

Server processing models developed from research results:

$$t_{latency} = \begin{cases} a_0 + a_1 N_{UE} + a_2 N_{UE}^2 + a_3 N_{UE}^3 + a_4 N_{UE}^4 & 1 \leq N_{UE} \leq 140 \\ b_0 + b_1 N_{UE} + b_2 N_{UE}^2 & 140 < N_{UE} \leq 960 \end{cases}$$

where $t_{latency}$ represents the average latency time to retrieve a Web object and N_{UE} , the number of User Equivalents. The specific parameters obtained through the fitting operations are summarized below in Tables 4.5 and 4.6.

Loner File Size	a_0	a_1	a_2	a_3	a_4
$s=1.143$	1.20610^{-2}	2.04610^{-3}	-4.15410^{-5}	3.42810^{-7}	4.55410^{-10}
$s=1.7145$	2.12210^{-2}	4.99210^{-4}	3.28210^{-5}	-4.82810^{-7}	4.12710^{-9}
$s=2.5$	5.03410^{-2}	2.44510^{-3}	-7.50410^{-5}	1.39010^{-6}	-1.63610^{-9}

Table 4.5 Parameters for polynomial fittings of the server models $1 \leq N_{UE} \leq 140$

Loner File Size	b_0	b_1	b_2
$s=1.143$	-2.177	2.16510^{-2}	-5.32710^{-7}
$s=1.7145$	-2.071	2.36810^{-2}	-1.71010^{-6}
$s=2.5$	-1.746	2.86910^{-2}	-3.99310^{-6}

Table 4.6 Parameters for polynomial fittings of the server models $140 < N_{UE} \leq 960$

Models representing the percentage of denied requests for $320 \leq N_{UE} \leq 960$:

$$errors/objects = c_0 + c_1 N_{UE} + c_2 N_{UE}^2 + c_3 N_{UE}^3 + c_4 N_{UE}^4$$

where *errors/objects* represents the average ratio of the number of “timeout” messages to the number of objects retrieved. The specific parameters obtained through the fitting operations are summarized below in Table 4.7.

Loner File Size	c_0	c_1	c_2	c_3	c_4
$s=1.143$	$-7.353 \cdot 10^{-4}$	$1.249 \cdot 10^{-5}$	$8.135 \cdot 10^{-8}$	$3.493 \cdot 10^{-12}$	$-1.527 \cdot 10^{-14}$
$s=1.7145$	$1.795 \cdot 10^{-4}$	$-1.707 \cdot 10^{-5}$	$2.422 \cdot 10^{-7}$	$-1.946 \cdot 10^{-10}$	$5.430 \cdot 10^{-14}$
$s=2.5$	$-5.245 \cdot 10^{-4}$	$-8.793 \cdot 10^{-6}$	$2.288 \cdot 10^{-7}$	$-1.604 \cdot 10^{-10}$	$3.984 \cdot 10^{-14}$

Table 4.7 Parameters for polynomial fittings of the percentage of denied requests

4.5 Integration and Validation of the Server Models

In this section, the server models are integrated into a simulation system and are validated. Since all simulation models involve approximation and abstractions, comparing measured and simulated results allowed us to verify the accuracy and the predictability of the *ns2* simulation model. To validate the server processing model, the average latencies and throughputs from a simulation system were compared to those of a comparable native single-node WWW server on a Linux platform. As this simulation system was meant to repeat simple real-world experiments, we attempted to configure it as realistically as possible. One single server is connected through a 10Mbps hub and responds to all requests from clients.

User Model

As we saw in Figure 4.1, the three UNIX computers are connected to the backbone through different 10 Mbps hubs. Each UNIX machine uses multithreading techniques to generate the desired number of User Equivalents making requests to the server. In this approach, each UE works on its own and makes requests to the server in parallel with other UE's. Assuming a network connect these UE's within a UNIX machine, the

bandwidth must be much higher than 10 Mbps. In our testbed, both requests from these UE's as well as responses from the server must go through the network interface of the UNIX machine, which has a bandwidth of only 10 Mbps. Thus, we model the connection between each UNIX and the 1Gbps backbone as a 10 Mbps duplex link.

Grouping UE's within each UNIX machine using a 100 Mbps duplex link, and then connecting the groups to the 1 Gbps backbone network via a 10 Mbps duplex link gives simulation results that are comparable to the SURGE measurements. However, connecting all users directly to the 1Gbps backbone through a 10 Mbps duplex link also gives matching results. This is because the queuing delay is only a trivial part of the overall delay compared to the server delay, which is further proved in Section 4.5.2. We therefore adopted the simpler architecture in Figure 4.13. To reflect the actual lab testbed for server stress-test experiments, users were divided into three groups to represent the three UNIX machines used in the SURGE tests. In other words, all users using one traffic input trace from SURGE represent one group, for a total of three groups. Loads were varied in our simulations by varying the number of users and the average requesting file sizes.

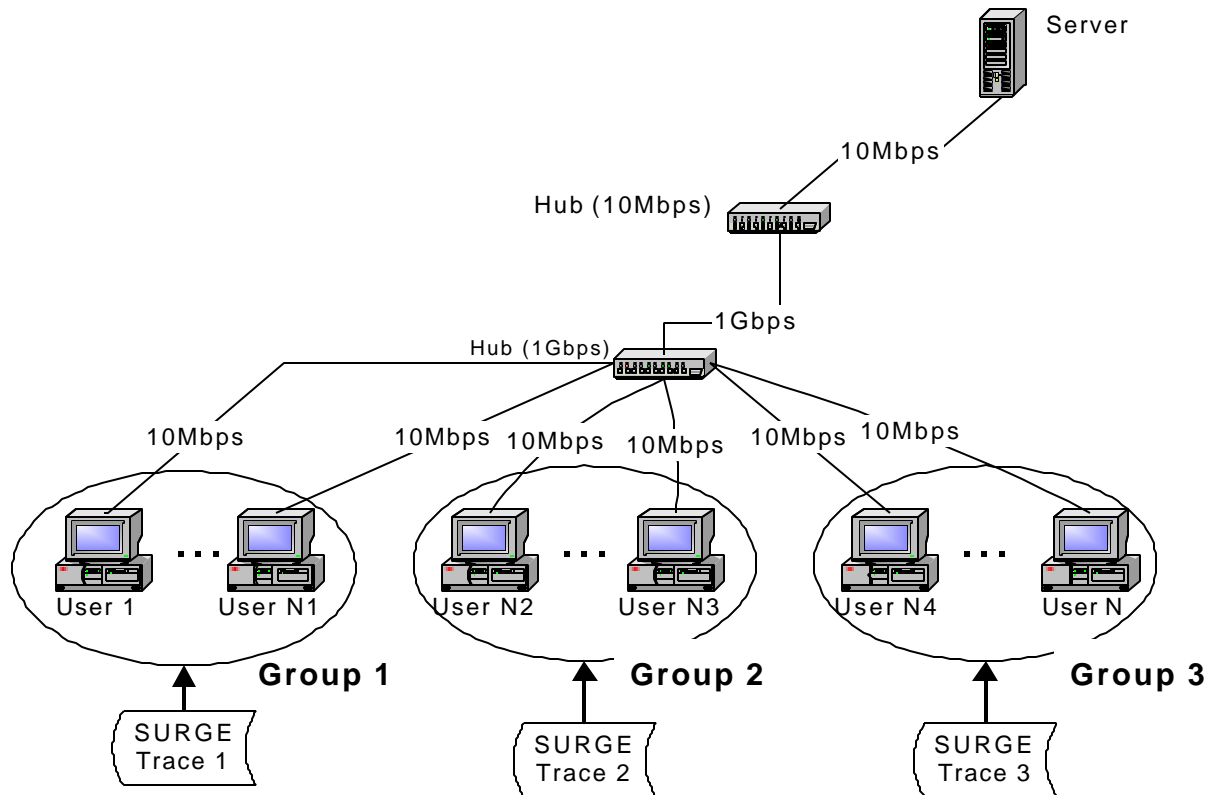


Figure 4.13 Simplified network architecture for server processing models

Traffic Input

Three traces from the SURGE workload generator were used as input to the *ns2* simulation system. These three traces were used in server stress tests in Section 4.3 and comprise of objects (a base HTML file and several embedded files make up a Web object), think-times between consequential object requests (after a user retrieves an object from the server, it waits a “think” time before making another request), and number of requests in a session (a user usually make requests for several objects from a particular Web server before moving onto another Web server). These traces follow probabilistic models and have been shown by Barford [Barford 1998] to be more realistic than those found in other existing benchmarks. Traces from ISP logs are difficult to maneuver. Since an increase in the load can occur on the basis of the number of uses accessing the server, the size and complexity of the Web pages, and the Web pages accessed, extending

an ISP trace to represent a higher load level is rather complicated. Using traces from SURGE makes it possible to perform extensive experiments, especially in simulating substantially higher traffic loads in a controlled fashion.

CPU Time Costs

In simulations of an Internet server, CPU time costs are usually modeled as a delay [Davison 2001]. Such models therefore assume that the transmission of other data is allowed in parallel. In real life, however, CPU time is an important resource and no connection requests are granted when this resource is exhausted. In SURGE, the error message “connection timed out” appears when there are no resources available. To account for this behavior, our simulation system adds the TCP socket receiving time-out value (180 microseconds) to the regular SURGE think-time model for the percentage of requests ending up with a “connection timed out” message. After an object is retrieved, a random number between 0 and 1 is generated and compared with the percentage of denied requests associated with the number of UE’s at the time being; if this random number falls below the percentage, 180 ms is added on top of the think-time as the new interval that the client stays idle.

HTTP Protocol Implementations

Our current simulation system ignores the low level details of the HTTP protocol, acts as a simple file server over the Internet, and ignores “if-modified” behaviors. “If-modified” behaviors refer to the conditional GET requests that the identified file be transferred only if it has been modified since the date given by the If-Modified-Since header sent out by the client. If the file has been modified since the “if-modified-since” date, the response is exactly as normal. Otherwise, the server only sends an “ok” status information and no file is transferred. Clients would read the file from its cache.

Persistent connections for HTTP/1.1 are supported. In other words, connections are left open between consecutive object transmissions for at least 15 seconds. Our simulation system mimics the disconnection process, which is initiated by the client when the last connection of a sequence of request is closed.

4.5.1 Integration of the Server Models

Figure 4.14 shows the flow chart of how the server models are implemented in the *ns2* simulation system.

1. A new session starts. As a request arrives at the server, if it is the first connection request to the server, the number of user equivalents N_{UE} increases by 1. If there is already at least one open connection to the server, send the request through the open connection. Otherwise, open a connection.
2. The server sends the HTML page to this client after a delay determined by a server processing model using N_{UE} as a reference.
3. Receiving the HTML page, the client decides the number of inlined images n_{image} and sends requests through open connections, if n_{image} is higher than the number of open connections N_{CONN} between the client and the server, the client opens up to 4 parallel connections to the server.
4. For each request, the server responds with the requested image file after a server delay. Again, this delay is determined by N_{UE} at that moment. These request/response processes continue until all the inlined images are withdrawn.
5. If this is the last object request in a session, the client closes all open connections to the server and N_{UE} decreases by 1.
6. A random number between 0 and 1 is generated and compared with the percentage of denied requests associated with N_{UE} for the time being. If the random number is less than the percentage, 180 ms adds to the “think time” model as the new “think time”.

7. The client stays idle for a “think time”. The connections leave open as defined by persistent-connection. During the whole period, if any connection stays idle for longer than 15 seconds which is the KeepAliveTimeout value, the server closes this connection.
8. After the “think time”, if all objects have been requested within a session, GOTO 1. Otherwise, the client sends request for another HTML page through one of the open connections, if none is open, the client opens one connection; GOTO 2.

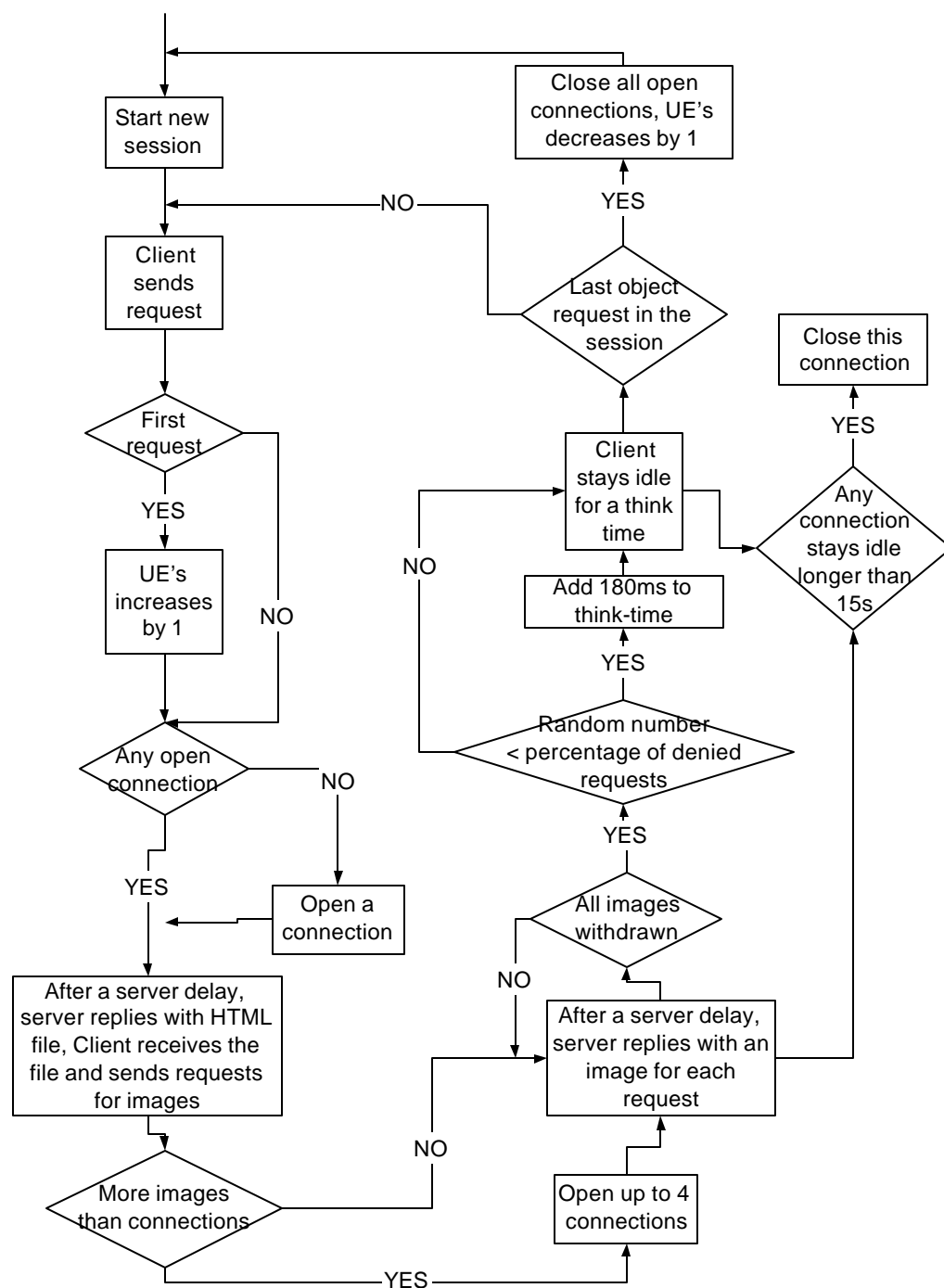


Figure 4.14 Flow chart of the server model implementation

4.5.2 Model Validation

We took the average of three simulation results for each UE number by varying the random seed in the simulator. Each simulation was run for 600 seconds, which is the same as in SURGE measurements. The average throughput of the server was measured in terms of the number of bytes processed and number of objects served (see Appendix C for details). In Figures 4.15 – 4.17, results from our simulations are compared with measurements made by SURGE. As can be seen from the Measured/Simulated ratio, the simulated results are comparable to the SURGE measurements. Our simulations less accurately predicted the lower load latency results, however, this is not serious as we are more interested in the higher latency regions.

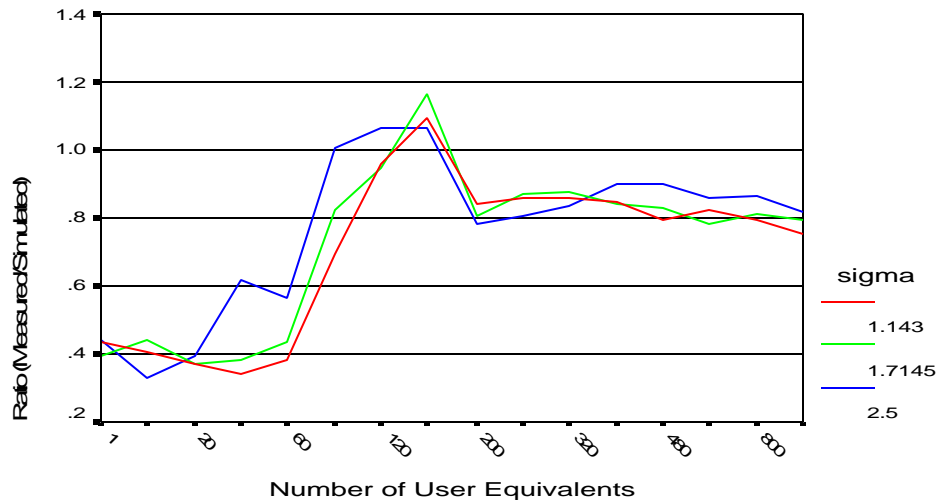


Figure 4.15 Comparisons of measured and simulated average latencies

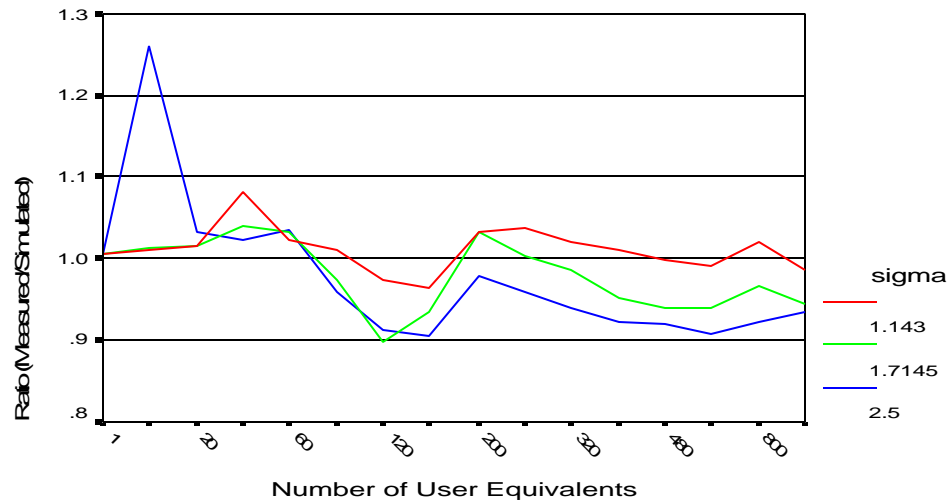


Figure 4.16 Comparisons of measured and simulated object throughputs

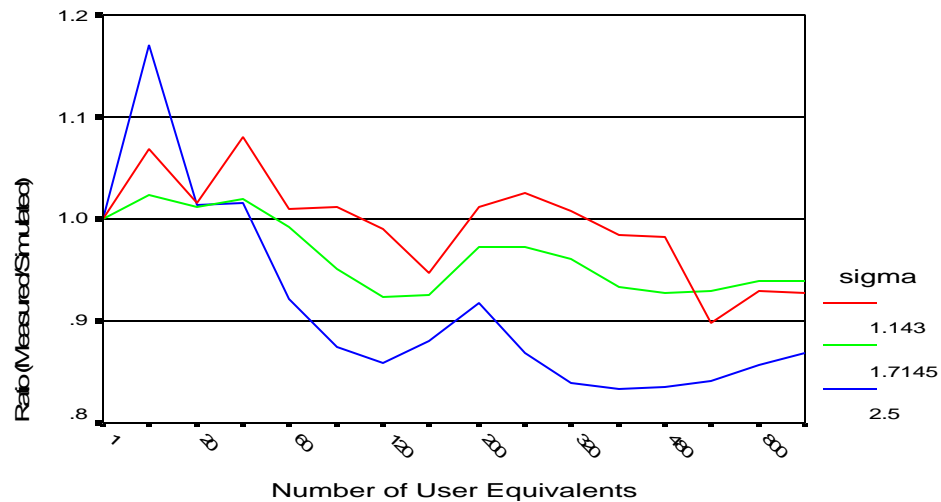


Figure 4.17 Comparisons of measured and simulated data throughputs

In order to observe the full behavior range of clustering technology, our server simulations were assumed to have an environment free of bandwidth constraints. However, our hardware resources were limited to a network speed of 10 Mbps and we were therefore unable to completely verify whether the server processing models are safe to apply in a 100 Mbps network environment. If the network delay is, in fact, a big part of the average latency, then the server processing models developed will consist of both

server and network delays. If not handled properly, these server models will then exaggerate the server delay, especially when the number of UE's is large. There are two ways to determine whether the network delay is an issue. One approach is to simulate the server processing model in a 100 Mbps network environment to determine the differences in average latencies compared to those in a 10 Mbps network. The second way is to assume zero server processing times, and develop a network delay model: a modified server processing model can then be constructed by subtracting the network delay from the original server processing model.

Figures 5.18 – 5.20 show the ratios of simulated results from a 100 Mbps network to those from a 10 Mbps network environment. Results from these two network environments are very close, which indirectly proves that the network delay is not significant in the server processing model. These results mean that it is safe to apply the server processing model in a 100 Mbps network environment.

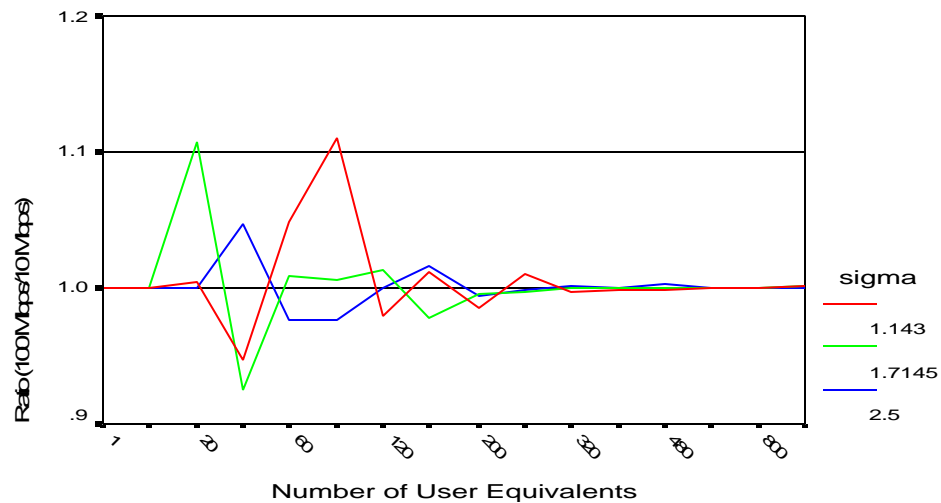


Figure 4.18 Comparisons of average latencies between 100 Mbps and 10 Mbps network environments

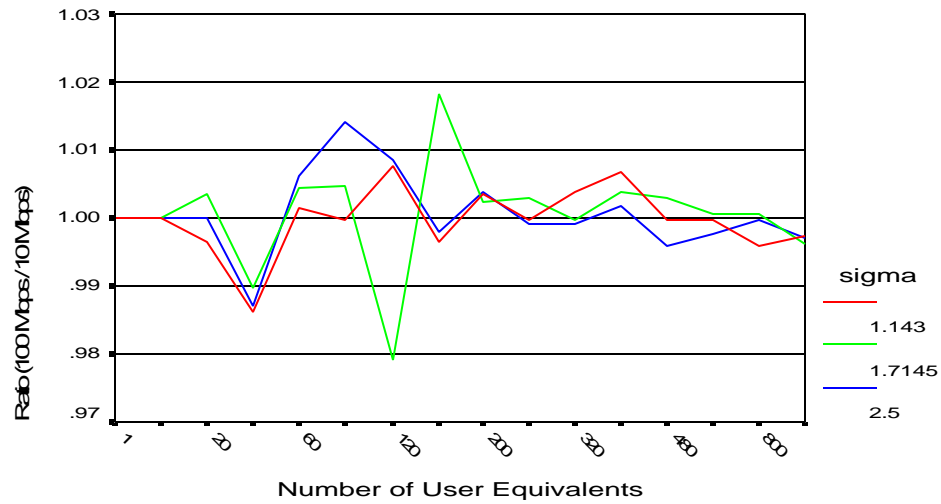


Figure 4.19 Comparisons of object throughputs between 100 Mbps and 10 Mbps network environments

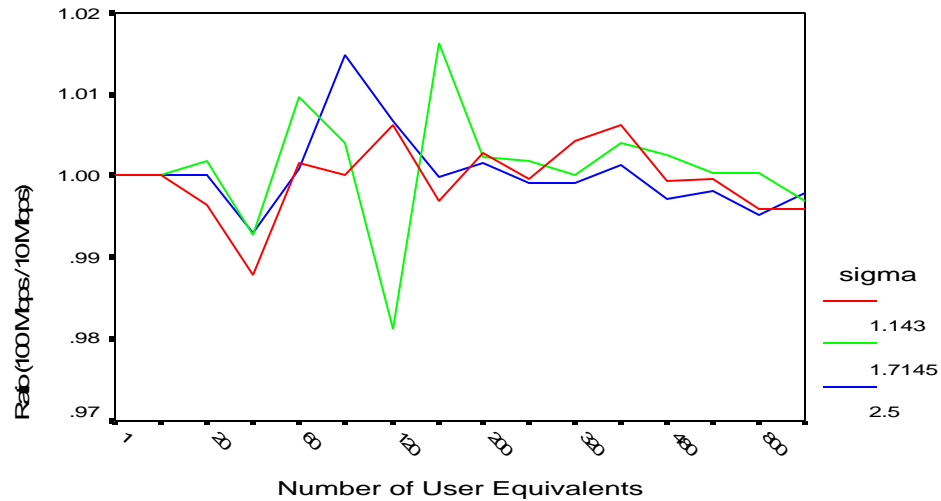


Figure 4.20 Comparisons of data throughputs between 100 Mbps and 10 Mbps network environments

In testing the second approach, we found that the revised server model significantly underestimated the server delays, as is shown in Figure 4.21. Assuming zero server delay, the time for the request/response process is dramatically shortened, which results in much more frequent object requests. Thus the network queuing delay is aggravated because of the increase in traffic.

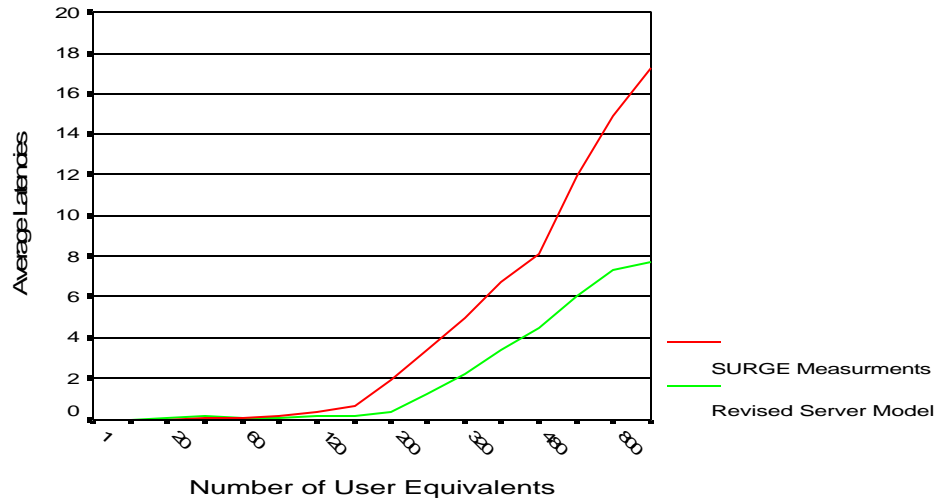


Figure 4.21 Results from revised server processing model and SURGE measurements $s = 1.143$

4.6 Summary

In this chapter, we described our SURGE-based stress-tests of a Linux server and used the results to build server models of the server that relate delay to the number of UE's in the system. These were integrated into an *ns2* simulation and further validated by comparing the latency predictions obtained from simulations with experimental results. These server models create the basis on which the clustering server architectures presented in Chapter 5 are constructed.

Chapter 5

Simulations and Results

The purpose of this research is to study server clustering technologies. One possible methodology is to set up a large testbed of real equipment and to experiment with real network traffic; however, building such testbeds is expensive and they are quite difficult to reconfigure. As the Internet and its servers change constantly, it is difficult for an individual group to acquire all resources needed to create a comprehensive and state-of-the-art networking environment. This limitation causes lack of standardization and renders reproduction of results by different sources unlikely. Another way to study server cluster technologies is by using a simulation system which, through formulas, mimics real world machines, networks and their characteristics. This method is abstract and does not involve real data transmission. It allows for easy maneuverability and uses very little resources. Another advantage of this simulation method is the exploration of possible future Internet architectures. We adopted this formula-based approach in our studies; however, in order to ensure a basis in “reality”, we based our server models on real measurements, as discussed in Chapter 4.

Our simulation system does not model all aspects of the Internet. Specifically, it does not model all traffic on relevant network links and does not attempt to perfectly match the actual host and network characteristics. In our simulation system the same network and host characteristics are given to each node and there are many fixed parameters in the models of the client, the network, the traffic and the server. In the real world, these values would be a function of ever changing conditions. The performance numbers shown in all figures and tables in this chapter can be useful for algorithm comparisons, but they should be taken with a grain of salt. The proposed network model is not comparable to the complexity of the real Internet.

5.1 Simulation Configurations

Figure 5.1 illustrates the server simulation model comprising of three components. The simulation system includes a clustered web-server model, a web-client model and a load balancer model. The user model, traffic input and HTTP/1.1 protocol implementations are the same as mentioned in Chapter 4.

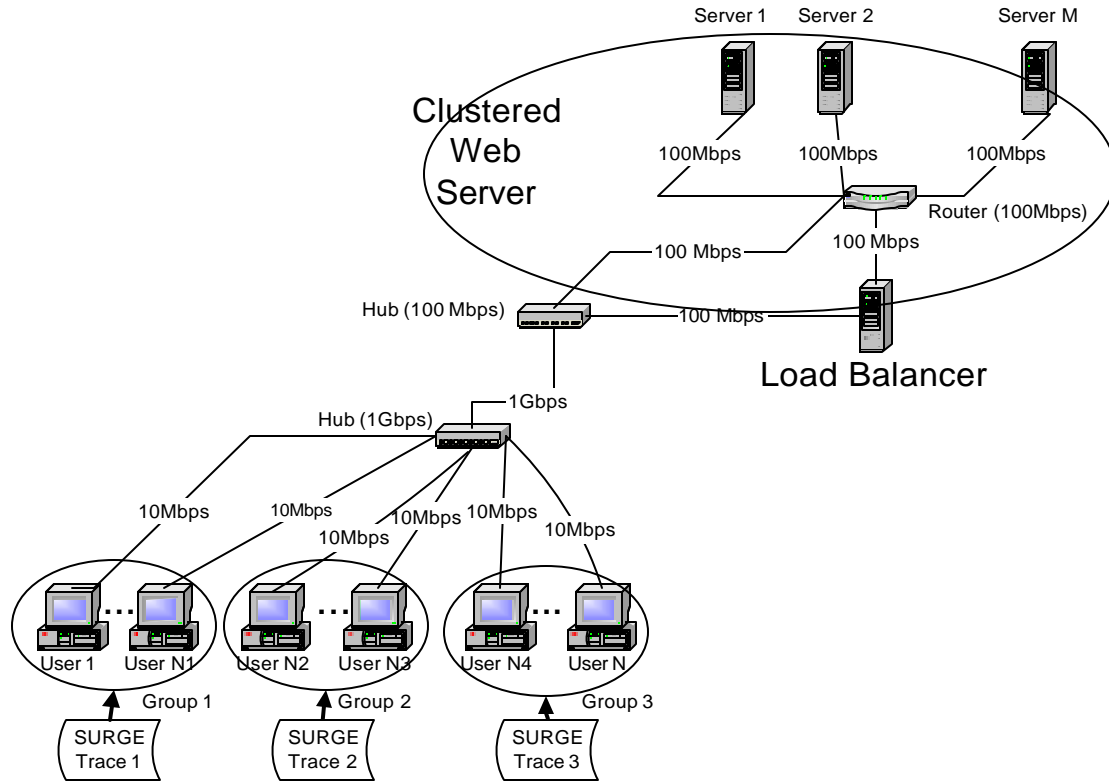


Figure 5.1 The clustered server simulation system architecture

5.1.1 Server Model

The web-server model contains a load balancer that functions as a “gateway” or a “switch” for server clusters. This component allows several server stations to appear as a single real server for processing HTTP requests. They are all interconnected with a 100Mbps router. Each server station has its CPU, central memory, hard disk and network interface and stores the same documents. The server processing model is based on the characteristic describing the average delay vs. number of user equivalents which was presented in Chapter 4. The number of user equivalents connecting to the server is used as a reference. In default, there are 4 server stations in a cluster.

5.1.2 Load Balancer Model

Both one-way (switch) and two-way (gateway) architectures are presented in this thesis. In the “gateway” architecture, both inbound and outbound traffic go through the load

balancer. In this architecture, the packets are rewritten at the TCP/IP level by the load balancer. Packet rewriting is based on the IP Network Address Translation (NAT) approach which is described in detail in Chapter 2. Our simulation system adds a 75 *ms* delay [Bestavros 1998] as a load balancing delay to account for each packet rewriting. Both the inbound and outbound packets experience rewriting delay. In the “switch” architecture, the outbound responses are sent directly to the users. In other words, the load balancer hands off the TCP connection to the selected server station. The routing from the load balancer to the server station can be achieved by rewriting the IP destination address or by forwarding the packet at the MAC level. Only the inbound packets experience extra delay as a result of packet rewriting in this approach, which becomes particularly important when serving large volumes of data.

Load balancers employ different routing strategies to balance the load across the system. In this thesis, five scheduling algorithms were studied. These five scheduling schemes represent three of the four classes of load balancing algorithms covered in Chapter 2. The only class we did not talk about is the content-based scheduling which is left for future work. The Round-Robin and Random policies do not count on any load information for load distribution, while the Least-Connection algorithm uses “instantaneous” connection information as a reference. The Round-Trip and Transmitted-Size methods calculate the statistics of the history state information. For definitions of those algorithms, refer to Chapter 2. In our simulations, we implemented these algorithms in the following manner:

1. Round-Robin (RR): A new connection request is assigned sequentially to the servers
2. Transmitted-Size (XSIZE): In this algorithm, the load balancer keeps track of the number of bytes transmitted from each server and chooses the one with the smallest count inside an averaging window. The default averaging window is 1 second. This value, if set too large, would cause the load balancer slow to respond to the sudden change in load. We consider 1 second reasonably small. To simplify

our experiments, the count is reset at the end of every averaging window. When two or more server stations have transmitted the same amount of bytes, the one with the least connections is chosen.

3. **Least-Connection (CONN):** The server with the least number of open connections is selected. A random choice is made among servers having the same number of connections. The interval for polling the server stations is 1 second. We assume this value the smaller the better, since it will give “instantaneous” state information of the server stations. To poll state information, an information port is installed on the server station. In typical implementation [Bryhni 2000], the HTTP server would periodically posts current load information to a load port that is independent of the HTTP process. The information port then replies to load queries from load balancer with a small UDP packet, implying an associated cost. We will discuss the impact of this polling interval on the load balancing later.
4. **Round-Trip (RT):** In this algorithm, the load balancer keeps track of the times between forwarding the requests to servers and receiving the first byte of the response. This information is then used to calculate the mean round-trip in an averaging window and to decide on the server assignment. This scheme is only realistic for the gateway architecture, since the traffic in both directions goes through the load balancer. For comparison, we also implemented this algorithm under the switch architecture. The time between load balancer’s forwarding the request to servers and the client’s receiving the first byte of the HTML page is recorded instead for the switch architecture. The mean round-trip value is reset at the end of the averaging window. As in the transmitted-size algorithm, one of the drawbacks of this scheme is the discontinuity of the averaging window. Both algorithms lose history information and depend solely on the number of concurrent open connections for scheduling decisions when the averaging window

is reset. The default averaging window is 1 second which is the same as in [Bryhni 2000]. The significance of this value will be discussed.

5. Random (RAN): The load balancer randomly assigns new connection requests to servers.

5.1.3 Network Model

In our network model, the load balancer connects the cluster's server stations to the 1 Gbps backbone network. The backbone network is connected to the load balancer by a 100 Mbps hub. Under the "switch" architecture, this hub also connects the server stations to the backbone network. Users are directly connected to the backbone via a 10 Mbps duplex link.

5.2 Simulation Results

In this section, we will present simulation results on latency, throughput, load balancing, and parameter alternating. Discussions on the implication of the results will follow each presentation. All simulation results are reported in detail in Appendix D.

5.2.1 Alternative Implementations

Prior to our study, we tried to duplicate results from [Bryhni 2000]. In his paper, Bryhni compared four scheduling algorithms: Round-Robin, Connections, Round-Trip and Xmitbyte (Transmitted-Size). Trace-driven simulations based on HTTP/1.0 protocol were carried out on a lab testbed. The implementations of the Round-Robin, Round-Trip and Xmitbyte schemes are exactly the same as ours. While in his Connections (denoted Bryhni-CONN from now on), we observed that when two or more server stations have the same number of active connections, the load balancer then chooses the one with the lowest server identifier [Bryhni 2000]. Compared to the Least-Connection scheme introduced earlier, the Bryhni-CONN method favours server stations with low server

identifiers to those with high numbers; whenever a request arrives to an empty system, the same server is chosen each time.

For comparisons, we implemented both the HTTP/1.0 and HTTP/1.1 protocols in our simulation. Figure 5.2 shows that the Bryhni-CONN-HTTP/1.0 method created an unbalanced load-distribution. This behaviour was also reported by Bryhni: strong load skew toward server stations with low number identifier [Bryhni 2000]. We extended Bryhni's results to a new version of the HTTP protocol. Bryhni-CONN-HTTP/1.1 also has some imbalance, although far less serious (Figure 5.3). These differences are due to the implementation of HTTP/1.1- Persistent Connection. Instead of closing each connection after the request is served, which results in high chance that a request arrives at an empty system, HTTP/1.1 leaves the connection open for consecutive requests. In this way, more open connections are maintained. Once the number of UE's crossed a threshold, the probability of a request arriving at an empty system becomes negligible. Also, the transitions in the number of open connections are smoother compared to those in HTTP/1.0. We are not able to compare the average latency or throughput between these two, since our server processing models were developed using HTTP/1.1-Persistent Connection.

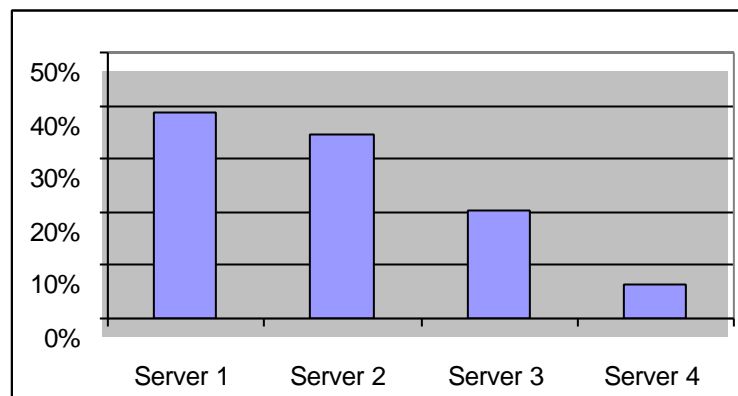


Figure 5.2 Percentage of bytes transferred (HTTP/1.0, Bryhni-CONN-G, $s = 1.143$, $N_{UE} = 640$)

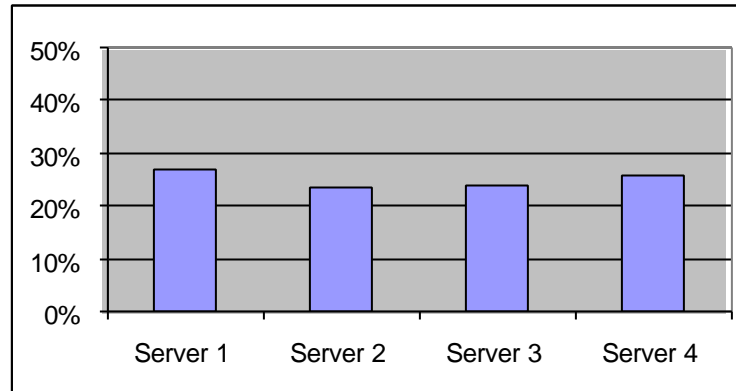


Figure 5.3 Percentage of bytes transferred (HTTP/1.1, Bryhni-CONN-G, $s = 1.143$, $N_{UE} = 640$)

We also implemented the Round-Robin, Round-trip and Transmitted-Byte algorithms in HTTP/1.0 and were able to replicate Bryhni's results [Bryhni 2000] in general conclusions. Round-Robin and Round-Trip methods achieve a fairly even load balancing in HTTP/1.0. Transmitted-Size schemes in both HTTP/1.0 and 1.1 have a skew toward lower numbered server stations (Figures 5.4 – 5.5).

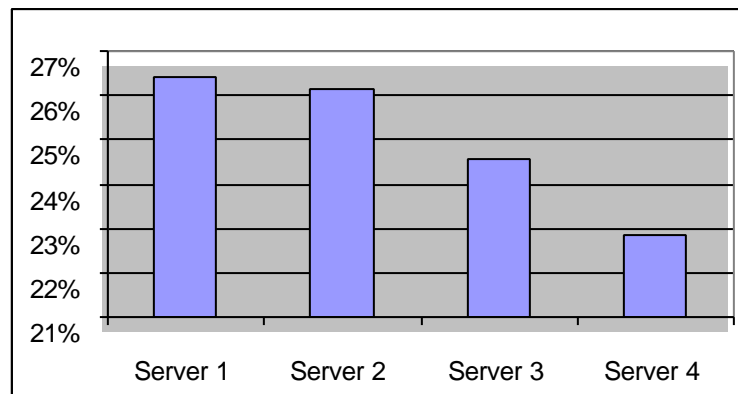


Figure 5.4 Percentage of bytes transferred (HTTP/1.0, XSIZE-S, $s = 1.143$, $N_{UE} = 640$)

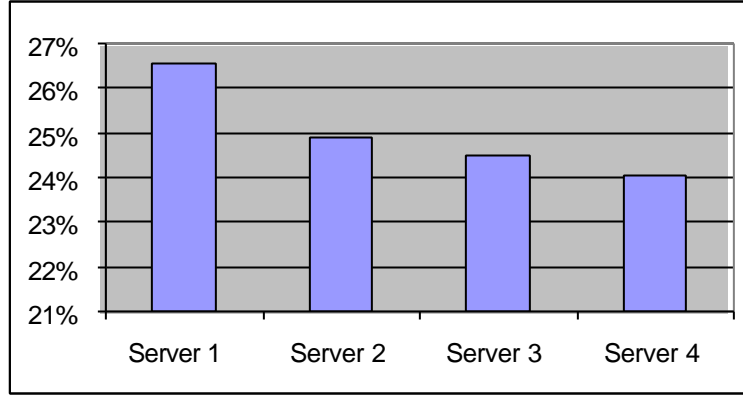


Figure 5.5 Percentage of bytes transferred (HTTP/1.1, XSIZE-S, $s = 1.143$, $N_{UE} = 640$)

Even though Bryhni-CONN creates the most serious imbalance in load distributions, Bryhni recommended Bryhni-CONN scheme, followed by Round-Robin on the basis of average response time and the algorithm complexity. He argued that slow responsiveness in the Round-Trip and Transmitted-Size algorithms leads to scheduling of request batches with good connection distribution but bad distribution with regard to response time. We will have further discussion of this argument later in this chapter.

The fact that we duplicated Bryhni's work is an indication of the validity of our simulator. We now proceed with the other simulation results.

5.2.2 Latencies and Throughputs

In Figures 5.6 – 5.8, the mean latencies and throughputs are compared in bytes/second and in number of objects served/second, for $s = 1.143$ and $N_{UE} = 640$ where N_{UE} is the number of User Equivalents. They show the typical behaviours of the different scheduling algorithms' behaviours (Refer to Appendix D for other results). As can be observed, the “switch” architecture outperforms the “gateway” architecture. As responses from servers do not go through the load balancer in the “switch” architecture, rewriting the packet is not needed, resulting in a smaller average response time. From the bar graph, we observe the following:

XSIZE \rightarrow RT \rightarrow RR \approx RAN \approx CONN

Slowest \rightarrow Average Latency \rightarrow Fastest

Smaller \rightarrow Throughputs \rightarrow Larger

The average latency for CONN is slightly less than those for RAN and RR (Table 5.1). We noticed that the response time distribution exhibits large skew and long tails for each of the distributions, with the distribution being strongly related to the characteristics of the files sizes on the server. Each sample size for response time distribution is way over 50000. The measurement of means is highly sensitive to the extreme outliers. Examining the stem-and-leaf plots for response time distributions in Appendix E reveals that most response times for RR and RAN spread between 0 and 4, while most of the response times for CONN center between 2 and 3. Some extreme outliers cancel out the effect of small values for RR and RAN. Therefore the mean is not accurate in predicting the scheduling algorithm giving the lower response time. A 90-percentile is more appropriate in this case. It represents the latency point where 90% of the object requests in the dataset have a lower latency than that. Table 5.1 summarizes the mean, 95% confidence intervals and 90-percentile with $s = 1.143$. Clearly, the Least-Connection scheme outperforms the other algorithms. The Round-Robin algorithm overall gives slightly lower 90-percentile of the response times than the Random scheme. A revision was made to the ordering of the response times:

XSIZE \rightarrow RT \rightarrow RAN \rightarrow RR \rightarrow CONN

This result is shown repetitively across the range of N_{UE} . We come up with the same order when s increases. Bryhni reported in his research [Bryhni 2000] that

“The Connections algorithm outperforms the algorithms that strive to estimate load based on history (Round-Trip and Transmitted-Size). On average, it gives slightly better response times than the round-robin algorithm.” “This is explained by the fact that many of the requests in the simulated traces are of the same kind; thus, it is

sufficient to count only connection state information (connections algorithm) or just perform round-robin scheduling.” [Bryhni 2000]

These findings concur with our results. Only measurements of the average latency were used in Bryhni’s study. We mentioned in Chapter 2 that the Round-Robin algorithm is effective when responses are of relatively-uniform size chunks and when computing powers of workstations are relatively comparable. This is exactly the case with our traffic. A sequence of non-optimal requests can create huge imbalances in loads among servers, which might render RR and RAN algorithms very inefficient.

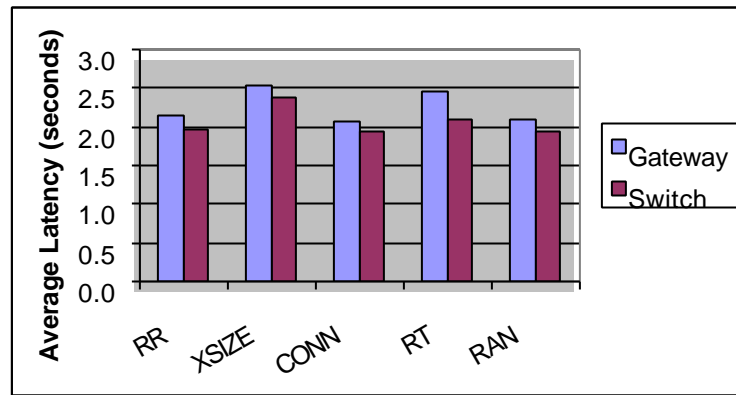
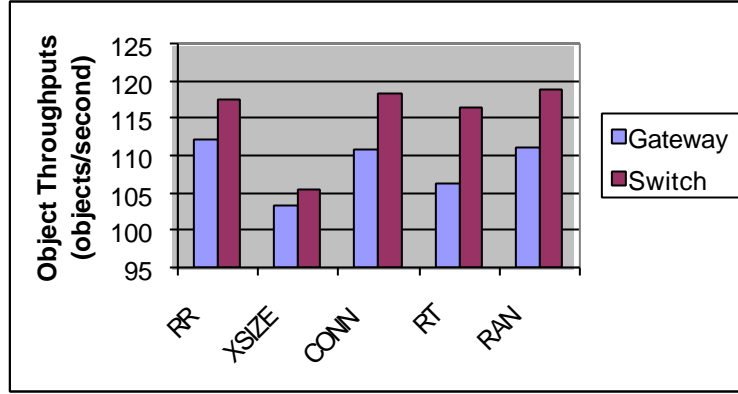
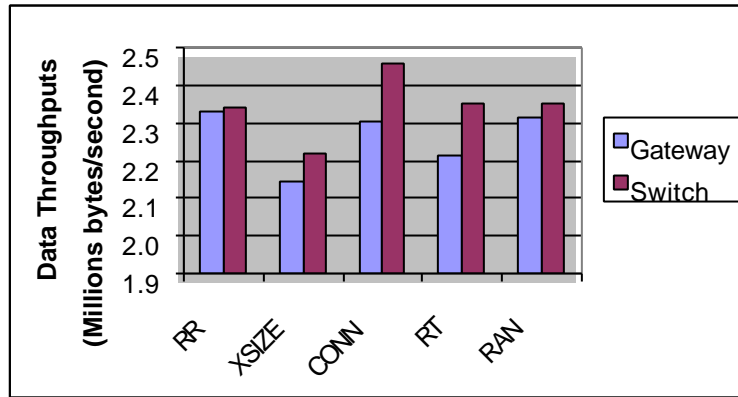


Figure 5.6 Average latency comparisons ($s = 1.143$, $N_{UE} = 640$)

	mean	95% Confidence Interval	90-percentile
RR-G	2.113	(2.060, 2.167)	3.780
XSIZE-G	2.504	(2.442, 2.567)	6.358
CONN-G	1.993	(1.943, 2.042)	2.591
RT-G	2.323	(2.272, 2.374)	6.244
RAN-G	2.149	(2.096, 2.201)	3.734

Table 5.1 Summary statistics ($s = 1.143$, $N_{UE} = 640$)

Figure 5.7 Object throughput comparisons ($s = 1.143$, $N_{UE} = 640$)Figure 5.8 Data throughput comparisons ($s = 1.143$, $N_{UE} = 640$)

The Transmitted-Size algorithm using the gateway architecture performs the worst out of the systems compared and we now demonstrate the efficiency of clustering technology by comparing its performance to that of the traditional single server. As we can see from Figures 5.9 – 5.11, Web clustering offers greatly superior performance than the traditional servers, both in terms of average latency and throughput. This result is more obvious under heavy workloads. When the number of user equivalents hits 140, at which point, the traditional server is overloaded, the difference in performance between clustered server and traditional server steadily increases with N_{UE} until the system throughput peaks at around 700 UE's. At this threshold, the system throughput of the clustered server is about four times the maximum throughput of a single server. Beyond this point, the system more or less maintains this maximum throughput. We conclude that

this clustered server is overloaded at 700 UE's. We consider this as a fair number, since the load threshold of a 4-node cluster is about 4 times that of a single server, the same with the peak throughput. These numbers mean that each server station in a cluster achieves its peak performance. As mentioned before, using several cheaper servers can achieve performance comparable to one high-performance model. Older models would not be wasted, the cost would be inexpensive compared to a high-end model and the server capacity can be extended by adding new stations as the workload increases.

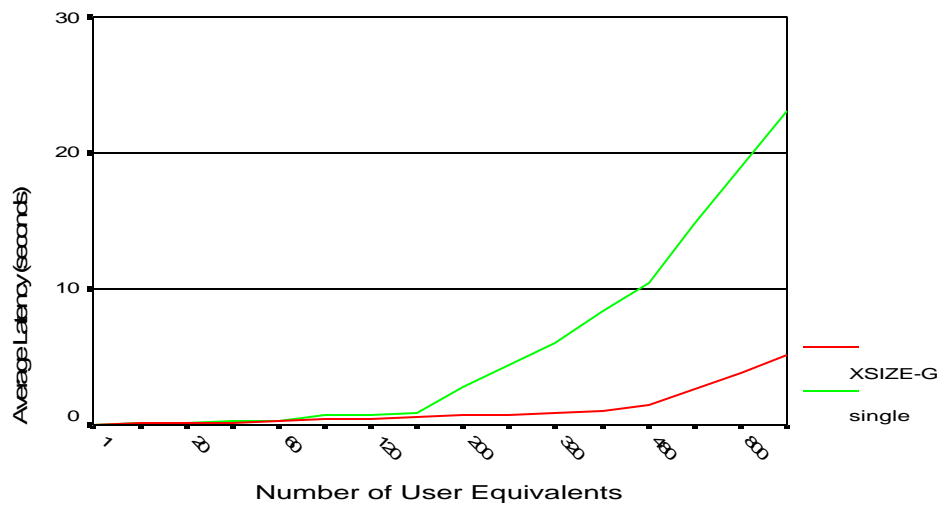


Figure 5.9 Average latencies for single server and XSIZE-G clustered server ($s = 1.143$)

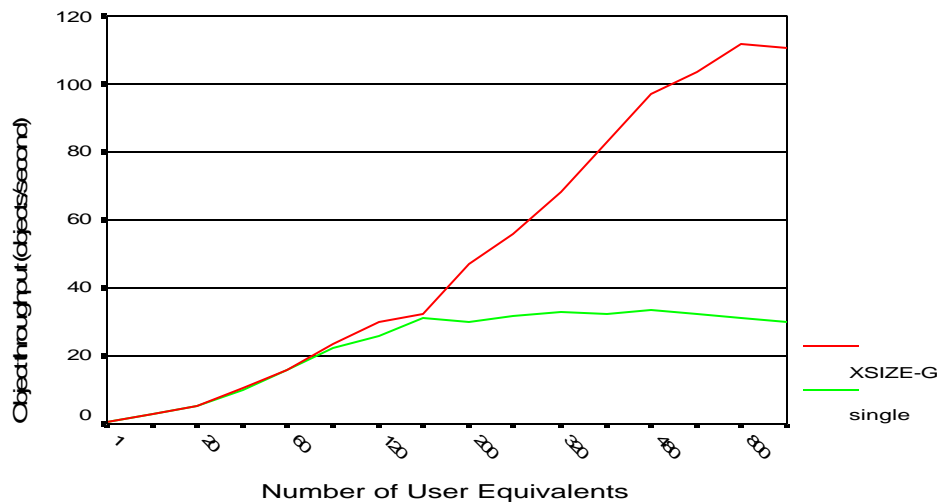


Figure 5.10 Object throughputs for single server and XSIZE-G clustered server ($s = 1.143$)

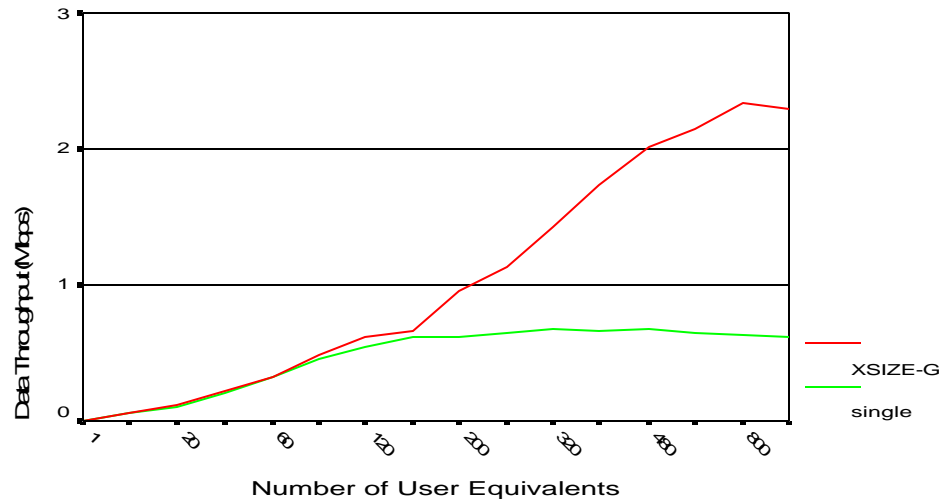


Figure 5.11 Data throughputs for single server and XSIZE-G clustered server ($s = 1.143$)

5.2.3 Load Balancing

The primary challenge of cluster architectures is to adequately balance load requests amongst servers. In an ideal situation, resources would be equally distributed and the load would be balanced, resulting in similar response times from all servers. Figure 5.12 show the histograms of loads under each of the five scheduling algorithms. The X-axis represents a range of concurrent open connections. The Y-axis represents the number of occurrences corresponding to the load. With an efficient scheduling scheme, all four servers would receive approximately the same number of connections, which means the distribution would concentrate around 150 – 200 connections for the case of 640 UE's. A “wider” spread of the distribution is an indication of an inferior policy, since the load is unevenly distributed, some servers get extremely large number of connections, while the rest receive very small number of connections.

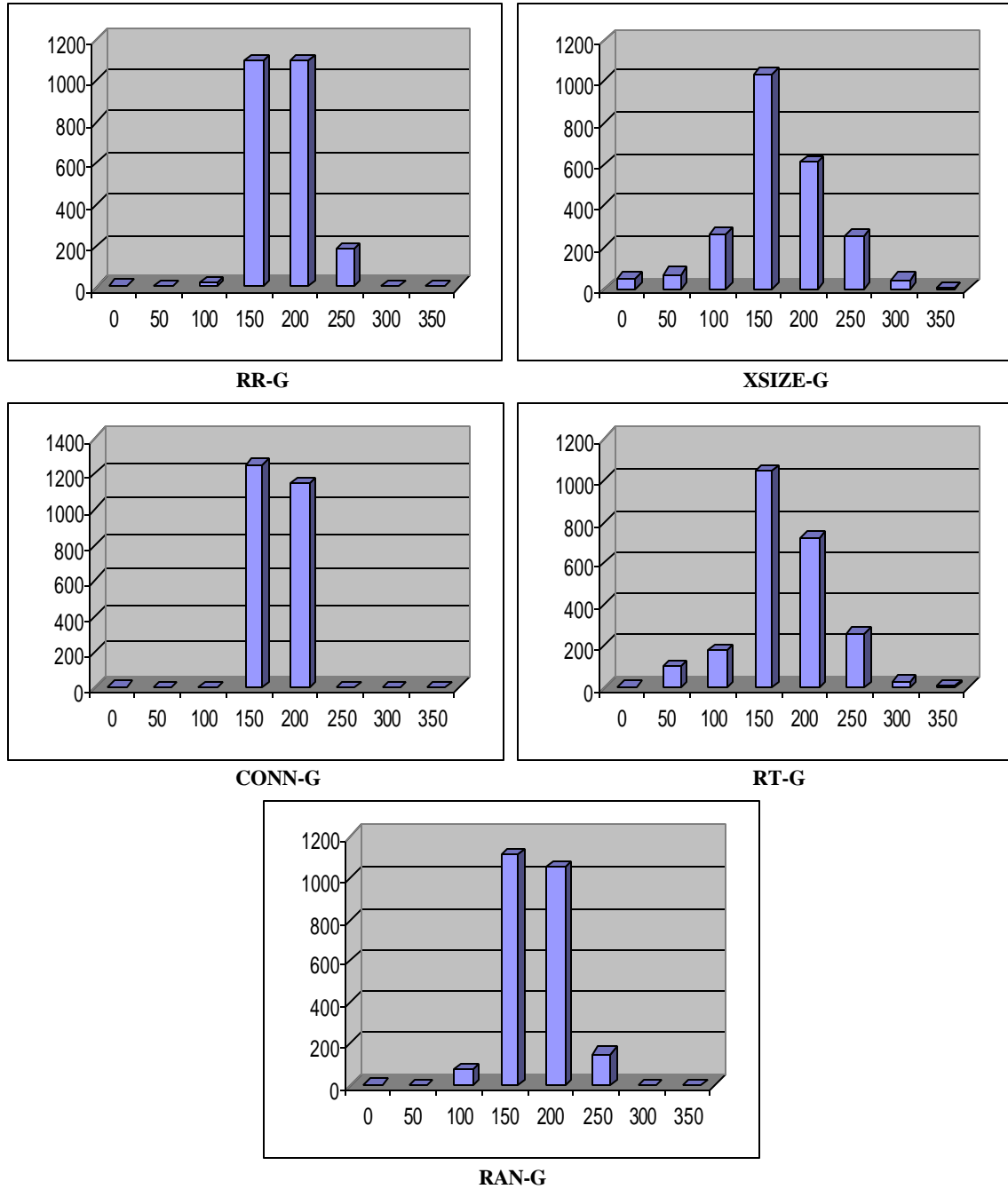


Figure 5.12 Rank of connections $s = 1.143$, $N_{UE} = 640$

Obviously, the CONN policy has the minimum spread and achieves the best load balancing. We order the load balancing efficiency of the five policies as:

XSIZE \rightarrow RT \rightarrow RAN \rightarrow RR \rightarrow CONN

Least Efficient \rightarrow Most Efficient

Tables 5.2 and 5.3 summary the mean, 10th percentile, 90th percentile and the load imbalance index defined by “ $1 + (90^{\text{th}} \text{ percentile} - 10^{\text{th}} \text{ percentile}) / 2 / \text{mean}$ ” [Aversa 2000]. The order of the imbalance index agrees with the order above, it also agrees with the order for the response time in Section 5.2.2. These agreements again prove that more efficient load balancing scheme results in lower response times.

	Connections			Imbalance Index
	Mean	10 th Percentile	90 th Percentile	
RR-G	155	119	197	1.25
XSIZE-G	151	89	218	1.43
CONN-G	149	136	162	1.09
RT-G	154	96	216	1.39
RAN-G	152	110	196	1.28

Table 5.2 Summary statistics for connections $s = 1.143$, $N_{UE} = 640$

	Connections			Imbalance Index
	Mean	10 th Percentile	90 th Percentile	
RR-G	160	133	188	1.17
XSIZE-G	158	69	245	1.56
CONN-G	156	146	167	1.07
RT-G	158	87	202	1.36
RAN-G	157	106	201	1.30

Table 5.3 Summary statistics for connections $s = 2.5$, $N_{UE} = 640$

However, it is not trivial work to collect information about the instantaneous number of concurrent open connections. Quite often, people use traffic traces recorded in the server logs to analyze the load balancing. Figure 5.13 illustrates the percentage of objects transferred from each server in the server cluster with 640 UE's and $s = 1.143$. This behavior is typical of clustered servers when the number of UE's is greater than 50. Comparing Figure 5.13 and 5.14 we see that the relative difference between the five algorithms remains the same in terms of objects transferred when s increases from 1.143 to 2.5. The load balancing distribution has different characteristics for UE's smaller than 50. When $N_{UE} < 50$, most requests arrive in an empty system, thus the load distributions

tend to be skewed toward the servers with lower server identifier, i.e. the servers with lower server identifier fulfill more requests. Figure 5.15 gives an example of the percentage of objects served by each server using the Transmitted-Size algorithm when there is only one user equivalent loading the system.

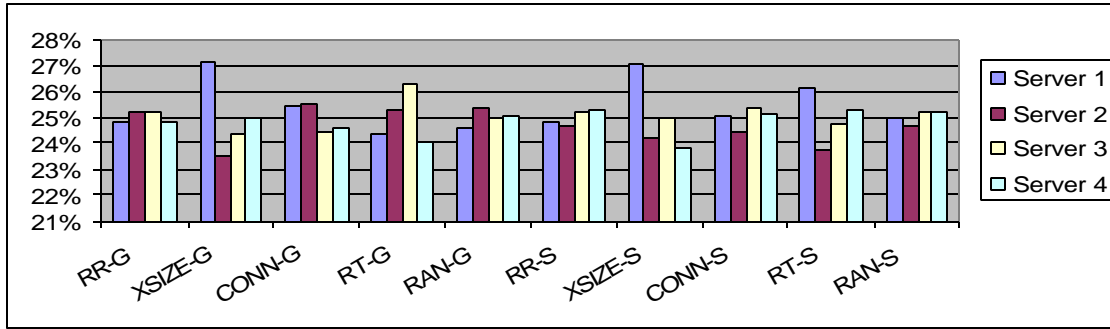


Figure 5.13 Percentage of objects transferred $s = 1.143$, $N_{UE} = 640$

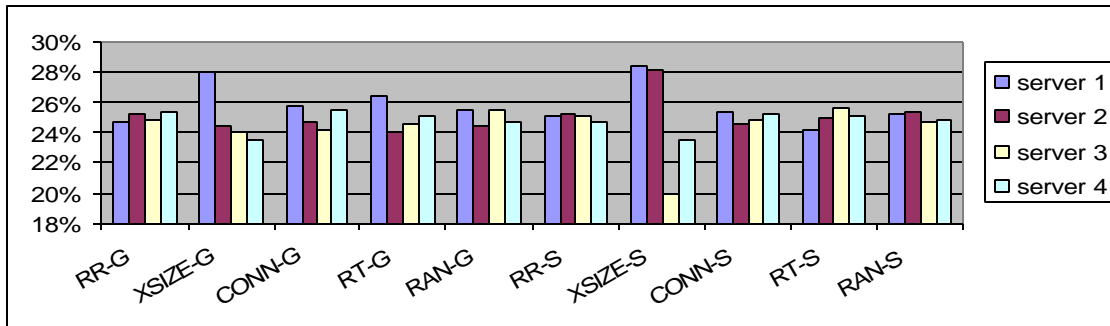


Figure 5.14 Percentage of objects transferred $s = 2.5$, $N_{UE} = 640$

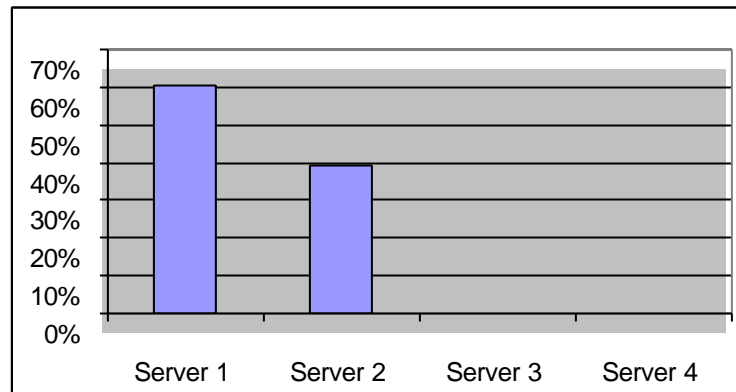
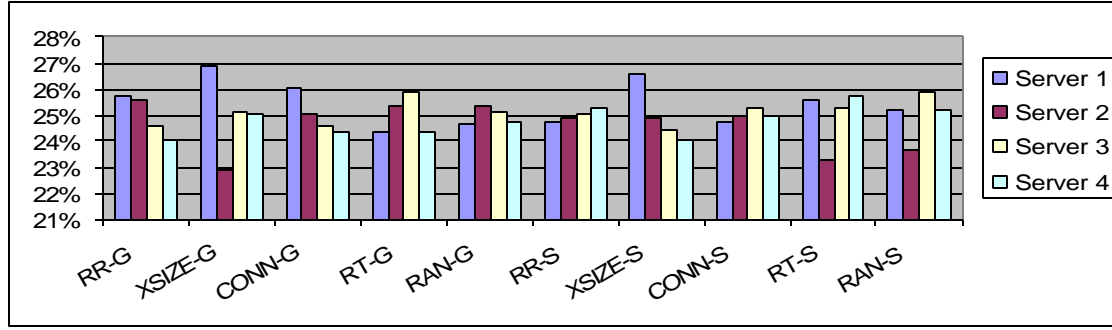
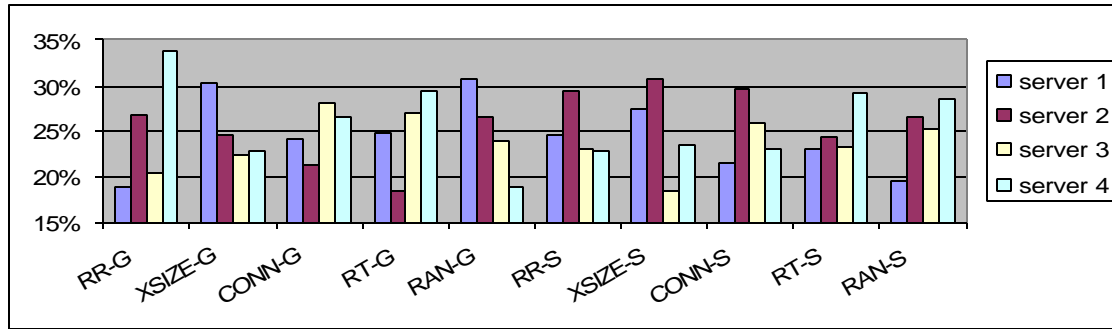


Figure 5.15 Percentage of objects served (XSIZE-G, $s = 1.143$, $N_{UE} = 1$)

Referring again to Figures 5.13 and 5.14, it is seen that the “gateway” and “switch” architectures produce results that are quite similar. Each server handles roughly the same number of objects for Round-Robin, Random and Least-Connection algorithms. However, this is not the case with Transmitted-Size and Round-Trip schemes. The uneven distributions of objects among servers results from the uneven load distributions. As in Chapter 4, we collected throughput results in bytes/second. We observe from Figure 5.16 and 5.17 that the load balancing measured in bytes transferred show more variability than that measured in objects processed. When S increases, this variability gets worse. From Figure 5.16, we can still claim that data throughput analysis leads to the same conclusions as object throughput analysis. We can hardly draw any conclusion from Figure 5.17. As mentioned in Chapter 4, the increase of S represents the increase usage of audio and video contents on the Internet. Increasing S dramatically increases the average file sizes. When S is small, each object carries approximately the same amount of bytes; bytes transferred measurement and objects served measurement present very similar load balancing results. A large S greatly affects bytes transferred measurements. In other words, the bytes transferred measurement is very dependant on the traffic pattern. As for objects served, it accurately reflects the number of open connections to the server, which has a direct influence on the server’s CPU and memory usage, i.e. workload. Along with our previous observation about distributions of the concurrent open connection and object served measurements, this result shows the superiority of using objects served over bytes transferred to accurately represent load balancing. Further discussions on the effect of average file size can be found in Section 5.2.4.

Figure 5.16 Percentage of bytes transferred $s = 1.143$, $N_{UE} = 640$ Figure 5.17 Percentage of bytes transferred $s = 2.5$, $N_{UE} = 640$

Round-Trip and Transmitted-Size perform poorly both in terms of average latency and load balancing. This observation is different from Bryhni's claim that the Round-Trip and Transmitted-Size algorithms leads to good connection distribution but bad distribution with regard to response time [Bryhni 2000]. In fact, the Transmitted-Size results in Bryhni's report also show a strong skew toward the server with low number identifier as we demonstrated in Section 5.2.1. But the skew in the case of Transmitted-Size is not as significant as that in Bryhni-CONN case which could be the reasoning that led him to his conclusions. Bryhni-CONN scheme favors the servers with low number identifiers, while we apply a random selection among servers with the same number of connections, which helps distribute the workloads.

5.2.4 The Effect of Average File Sizes

In Chapter 4, we developed three server processing models with different s 's to represent increasing multimedia usage on the Internet. File size has a negative impact on latency, which was expected. With the increase of s , the object throughput decreases, the data throughput increases (Figures 5.18 – 5.20), while the loads distribution among the servers stays about the same in terms of objects and varies more in terms of bytes (Figure 5.21).

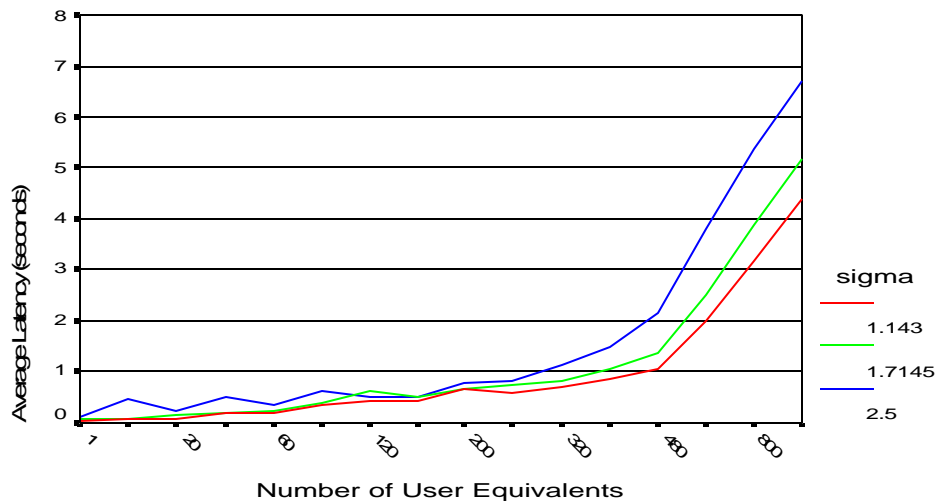


Figure 5.18 Comparison of average latencies with different s 's (RR-S)

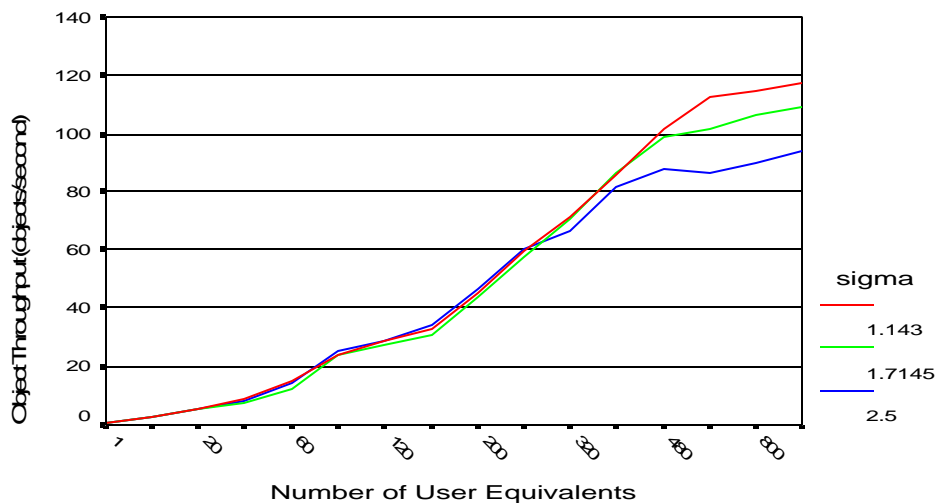
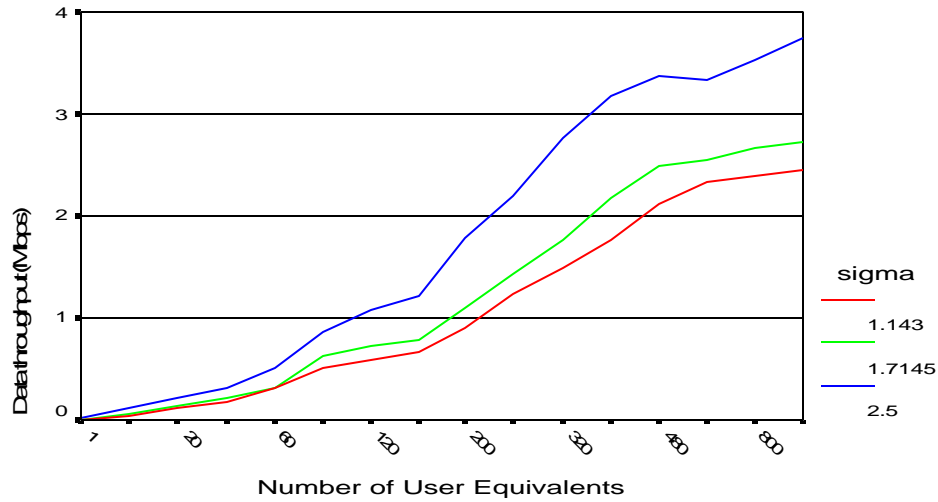
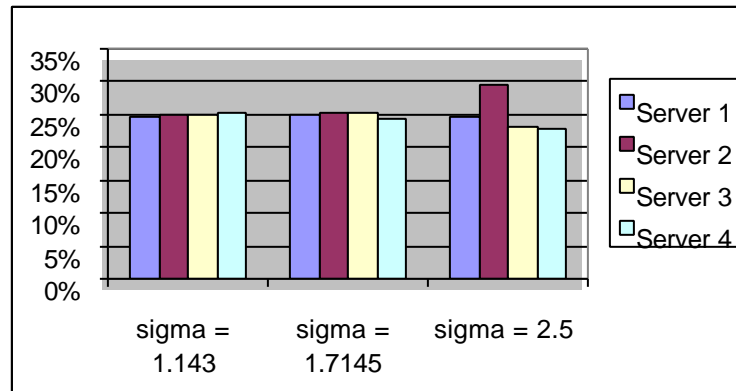


Figure 5.19 Comparison of object throughputs with different s 's (RR-S)

Figure 5.20 Comparison of data throughputs with different σ 's (RR-S)Figure 5.21 Load balancing in term of bytes with different average file sizes (RR-S, $N_{UE} = 640$)

5.2.5 The Effect of Number of Servers in a Cluster

The number of server stations in a cluster directly influences the number of UE's attached to the servers, which controls the server delay. To illustrate this point, let's take "gateway" architecture with the Round-Robin algorithm as an example. Figure 5.22 shows that increasing the number of servers in a cluster dramatically decreases the average latency under heavy loads. Figures 5.23 – 5.24 illustrate that the object throughput and data throughput show very similar trends. As can be expected, the asymptote giving the maximum throughput increases with the number of servers in a cluster. Figures 5.25 – 5.26 help us to clearly see the efficiency of increasing the number

of servers in a cluster, where average latency ratio R_t and the object throughput ratio R_o are respectively defined as:

$$R_t = t_{1-server} / t_{N-server}$$

$$R_o = O_{N-server} / O_{1-server}$$

where $t_{N-server}$ is the average latency and $O_{N-server}$ is the object throughput of a cluster with N servers.

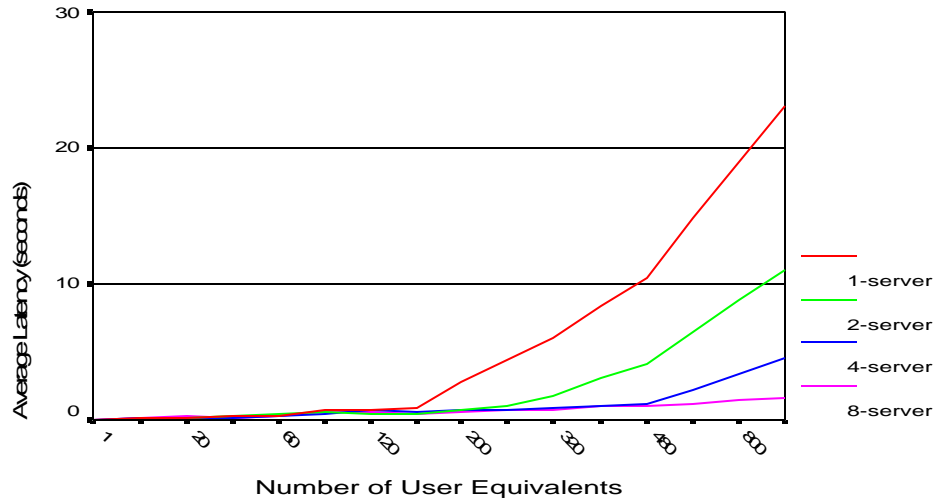


Figure 5.22 Effect of varying the number of servers on the average latency (RR-G, $s = 1.143$)

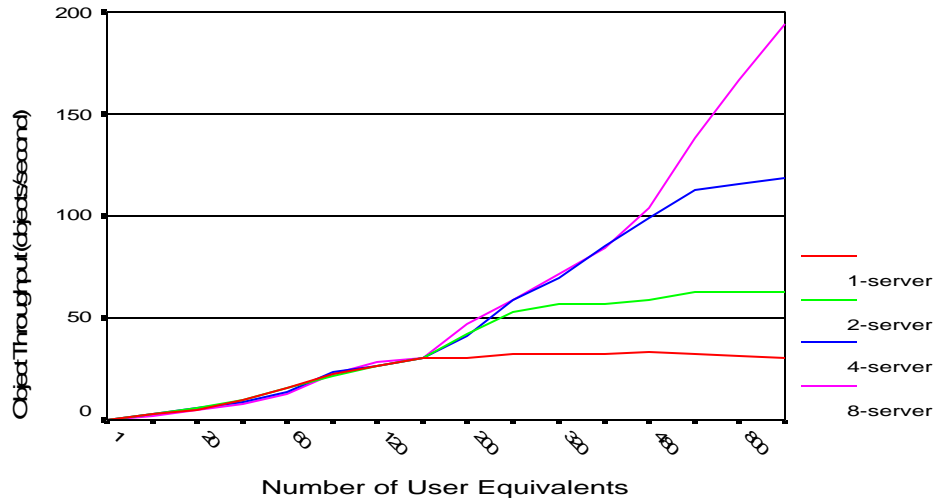


Figure 5.23 Effect of varying the number of servers on the object throughput (RR-G, $s = 1.143$)

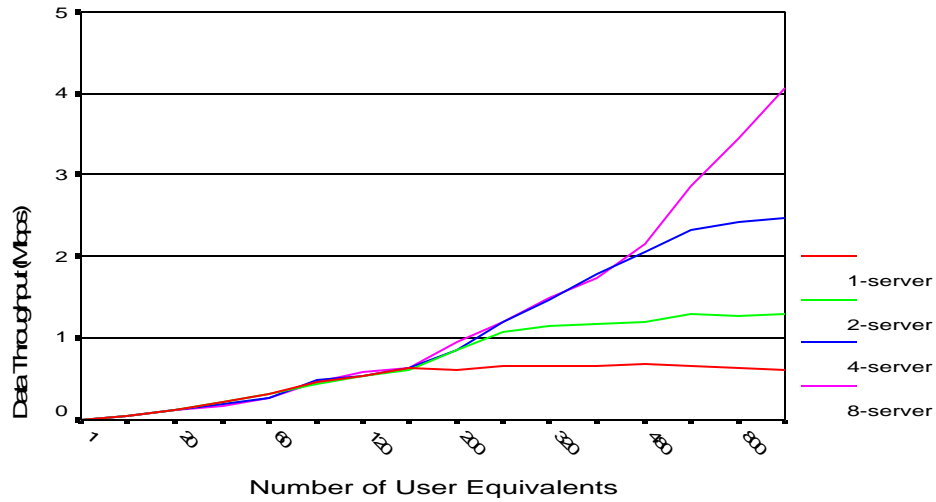
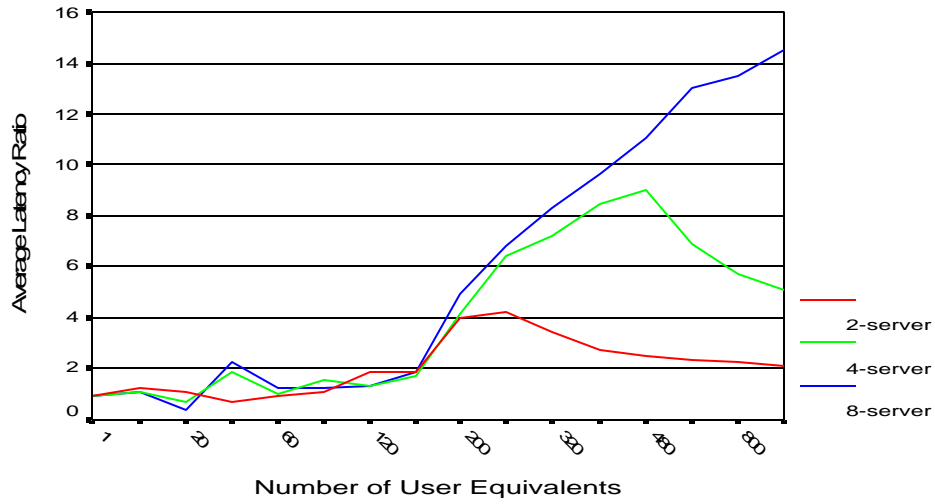
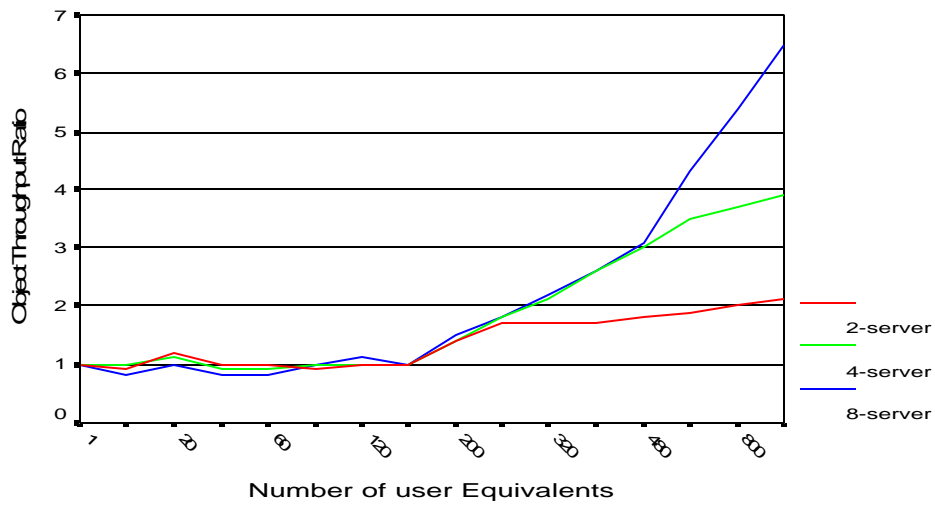


Figure 5.24 Effect of varying the number of servers on the data throughput (RR-G, $s = 1.143$)

When the number of User Equivalents is less than 140, there is not much saving in the average latency or increase in the throughput. As the clustered server is under-loaded, a request most likely arrives to an empty system. We found that 2-node cluster is overloaded when the number of UE's reaches 280. Figure 5.23 indicates that the peak system throughput holds steady once this threshold has been reached. The throughput for a cluster with 4 nodes continues to increase and reach its threshold at about 600 UE's. In our simulations, the throughput of the 8-node cluster continue to increase beyond the 4-node threshold. As we were unable to simulate a number of user equivalents higher than 960, we could not determine the 8-node cluster's load threshold. In our simulations, we found that when a cluster with N servers is under heavy loads, it can achieve an average latency $1/N$ of that from a single server. As for throughput, it was found to be almost N times higher than that of the traditional server.

Figure 5.25 Average latency ratios R_l (RR-G, $s = 1.143$)Figure 5.26 Object throughput ratios R_o (RR-G, $s = 1.143$)

5.2.6 The Effect of Network Performance Parameters

At the beginning of this chapter, we mentioned that our server processing model can be applied in a 10 Mbps or a 100 Mbps network environment, and that we wanted to study the full behaviour range of clustered servers: i.e., the region where clustered server is the bottleneck as opposed to the network. In this section, we study the opposite situation in which the network becomes a bottleneck. As shown in Figure 5.27, the connection between load balancer and the 1 Gbps backbone network is changed to 10 Mbps. The

same change applies to the connection between the cluster router and the backbone network. Under heavy load, the average latencies for 10 Mbps network are much higher than those for 100 Mbps (Figure 5.28). Network bandwidth limits the data throughput as illustrated in Figure 5.29.

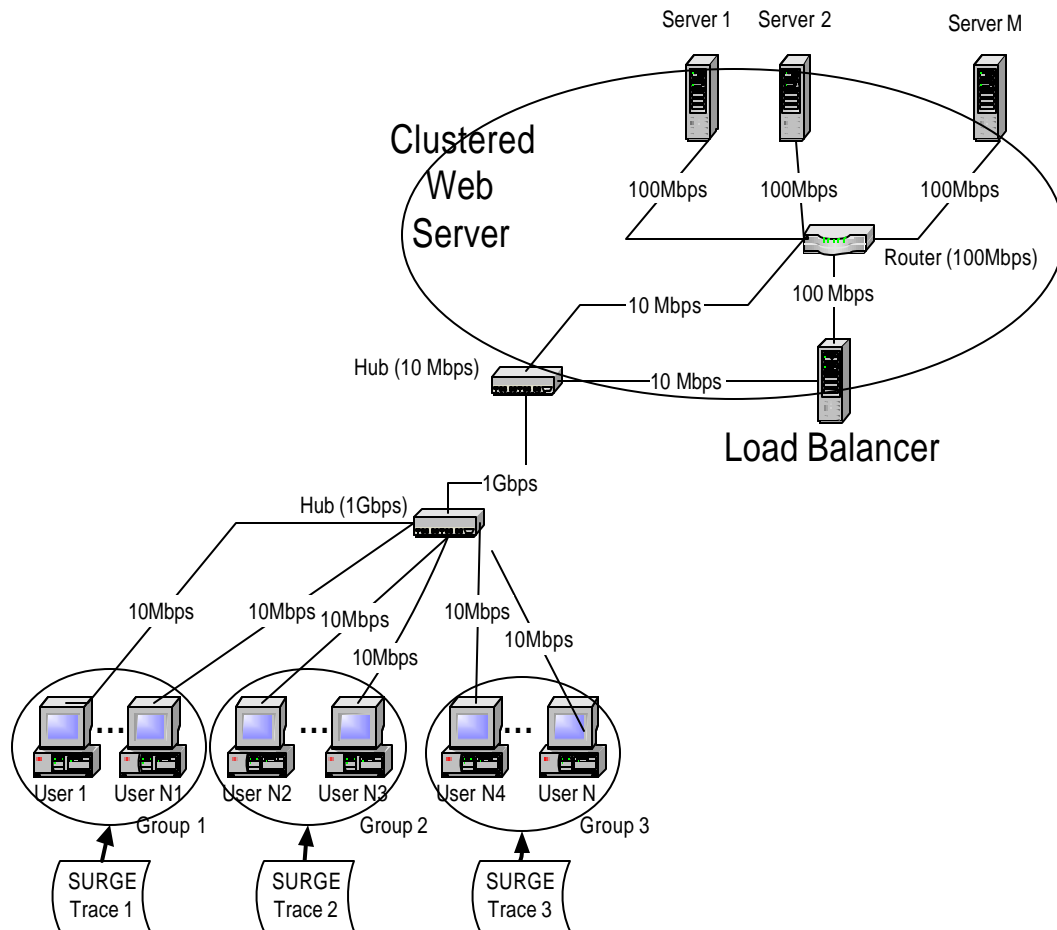


Figure 5.27 The alternative simulation architecture

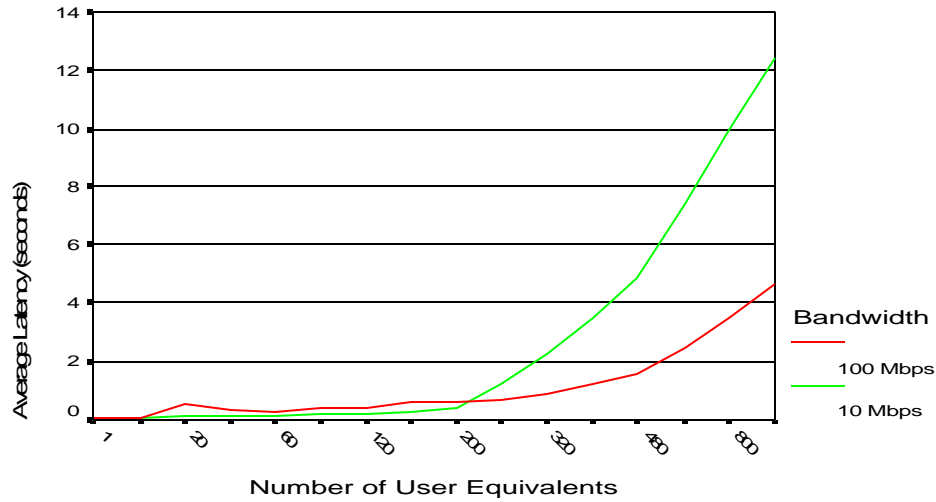


Figure 5.28 Comparison of average latencies under two different network environments (CONN-G, $s = 1.143$)

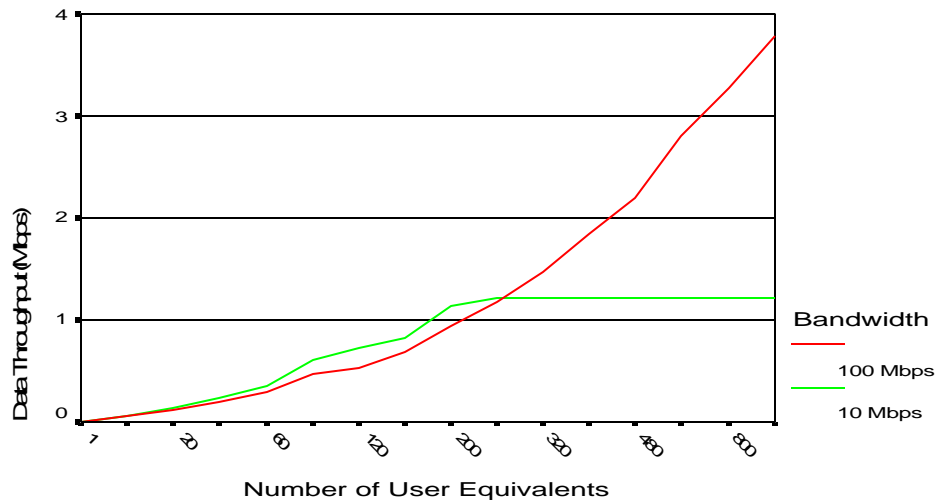


Figure 5.29 Comparison of data throughputs under two different network environments (CONN-G, $s = 1.143$)

5.2.7 The Impact of Averaging Windows

In order to fully understand the behavior of the servers we study, we found it crucial to vary some parameters. The duration of the averaging window, for example, significantly affects the performance in the Transmitted-Size, Least-Connection and Round-Trip policies. In general, bigger window sizes cause the history information to be averaged over a longer period of time, while smaller ones cause the load balancer to recalculate balancing criteria more frequently which can provide rapid response to changing

conditions. As expected, performance of all policies tends to improve for higher values of the frequency of load information broadcasts. This phenomenon could be attributed to better load balancing with shorter window size. Unfortunately, however, there is a cost involved with polling the server stations. Using relatively short update periods places a heavy burden on the load monitor since it has to poll individual server stations more frequently. We did not simulate this polling process in present simulation system.

However, it could be accommodated in future work. At this point, we can not comment on the effect of this cost on the performance.

From Figures 5.30 and 5.31, we can see that when the averaging window size reduces to 0.1s for the Transmitted-Size scheme, the average latency decreases significantly. Table 5.4 confirms that the performance level dramatically increases. The histogram of the ranks of connections (Figure 5.32) and imbalance index (Table 5.5) illustrate that the spread of the load distribution greatly shrinks. Figure 5.33 illustrates that the XSIZE policy with averaging window of 0.1s achieves even object throughput distributions.

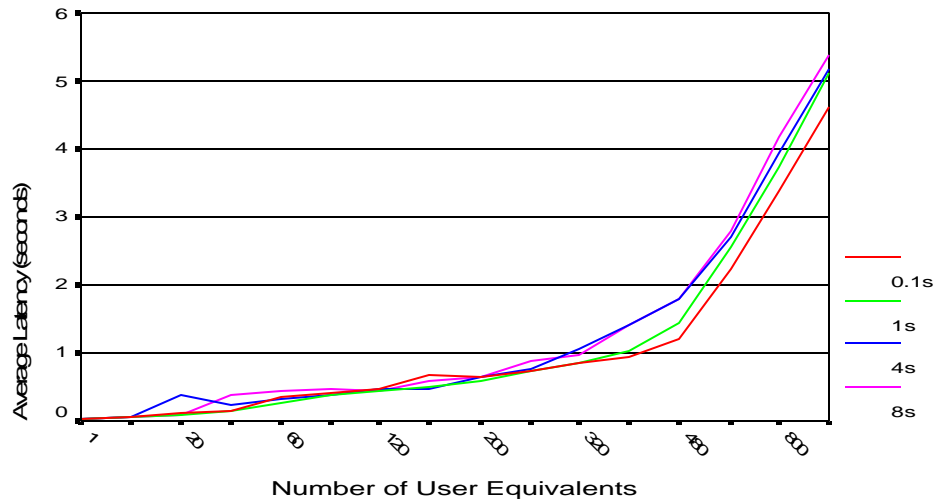


Figure 5.30 Comparison of average latencies with different averaging windows (XSIZE-G, $s = 1.143$)

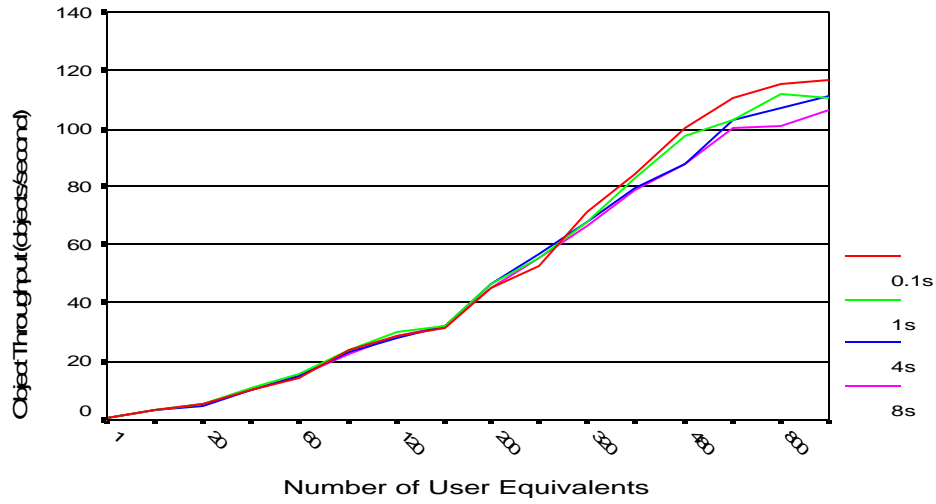


Figure 5.31 Comparison of object throughputs with different averaging windows (XSIZE-G, $s = 1.143$)

	mean	95% Confidence Interval	90-percentile
RR-G	2.113	(2.060, 2.167)	3.780
XSIZE-G-1s	2.504	(2.442, 2.567)	6.358
XSIZE-G-0.1s	2.197	(2.141, 2.254)	4.693
CONN-G	1.993	(1.943, 2.042)	2.591
RAN-G	2.149	(2.096, 2.201)	3.734

Table 5.4 Summary statistics ($s = 1.143$, $N_{UE} = 640$)

	Connections			Imbalance Index
	Mean	10 th Percentile	90 th Percentile	
XSIZE-G-1s	151	89	218	1.43
XSIZE-G-0.1s	153	109	205	1.31
CONN-G	149	136	162	1.09
RAN-G	152	110	196	1.28

Table 5.5 Summary statistics for connections XSIZE-G, $s = 1.143$, $N_{UE} = 640$

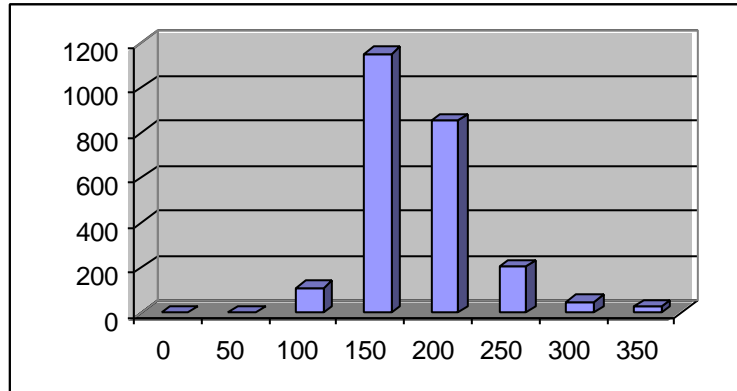


Figure 5.32 Rank of connections under XSIZE-G policy $s = 1.143$, $N_{UE} = 640$, averaging window = 0.1s

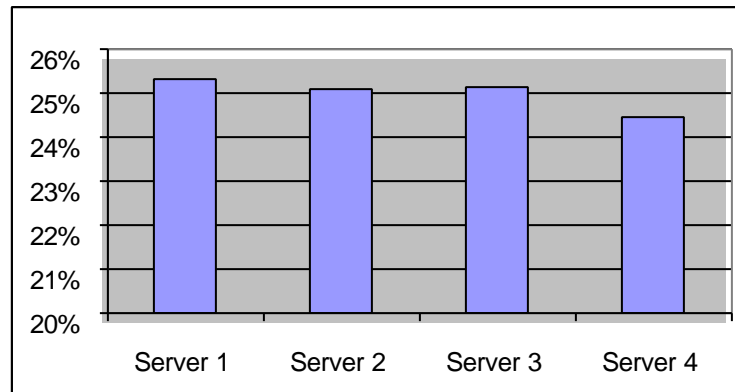


Figure 5.33 Percentage of objects served (XSIZE-G, $s = 1.143$, $N_{UE} = 640$, averaging window = 0.1s)

We found from Figures 5.34 – 5.35 that a threshold $\geq 8s$ decreases the performance of Least-Connection algorithm. A threshold between 0.1s to 4s performs equally well.

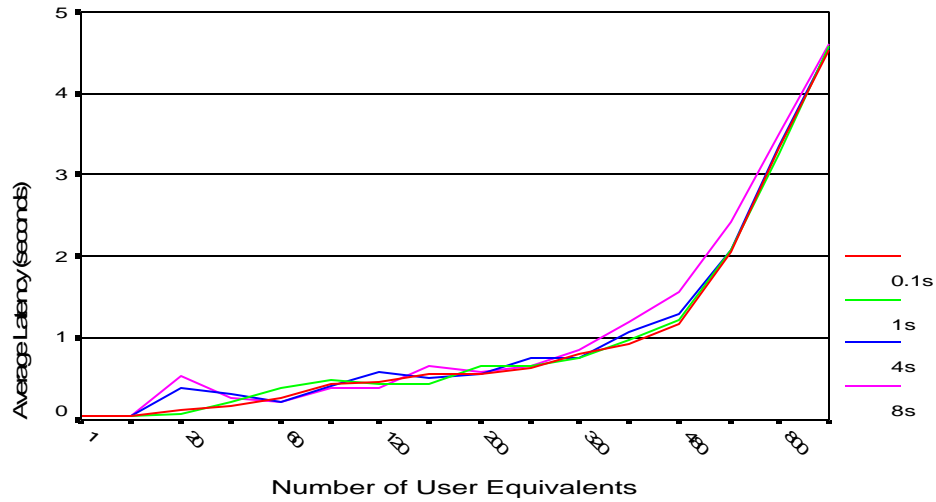


Figure 5.34 Comparison of average latencies with different averaging windows (CONN-G, $s = 1.143$)

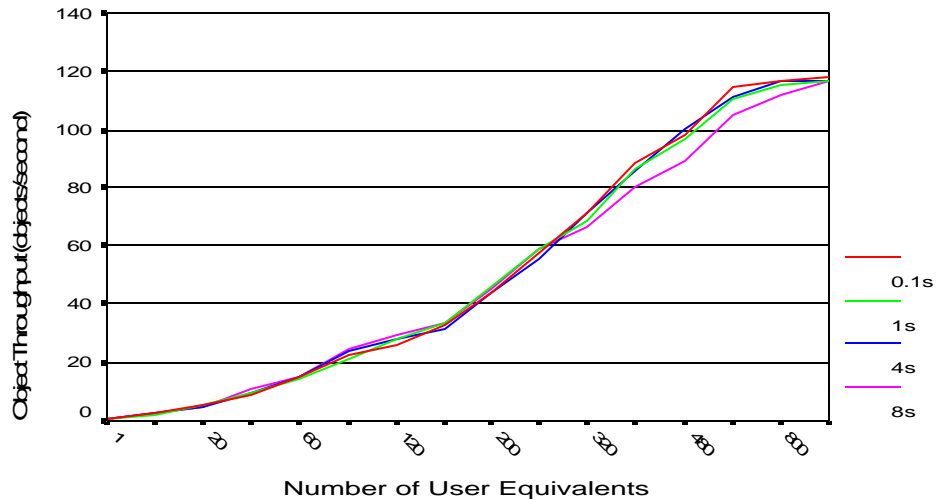


Figure 5.35 Comparison of object throughputs with different averaging windows (CONN-G, $s = 1.143$)

Figures 5.36 – 5.37 compare the average latencies and throughputs for Round-Trip algorithms with different averaging window size. An even better performance boost was found for the Round-Trip algorithm with 0.1s averaging window than those for the Transmitted-Size scheme with 0.1s threshold. We conclude that with sufficiently small averaging windows, Round-Trip algorithms can achieve performance level comparable to that of the Least-Connection schemes (Tables 5.6 – 5.7). From Figure 5.38, we see that the load distribution is more concentrated when the averaging window is smaller. The

Round-Trip policy keeps track of the times between forwarding the requests to servers and receiving the first byte of response. When the averaging window is small enough, the load balancer is effectively sending dynamic probes to all servers and selecting the first to reply, which is exactly dynamic or run-time scheduling scheme introduced in Chapter 2. This experiment shows that the Round-Trip method with small enough averaging window is definitely better than Transmitted-Size policy, even though they both belong to the statistical algorithms. The Round-Trip policy can only be implemented in “Gateway” architecture.

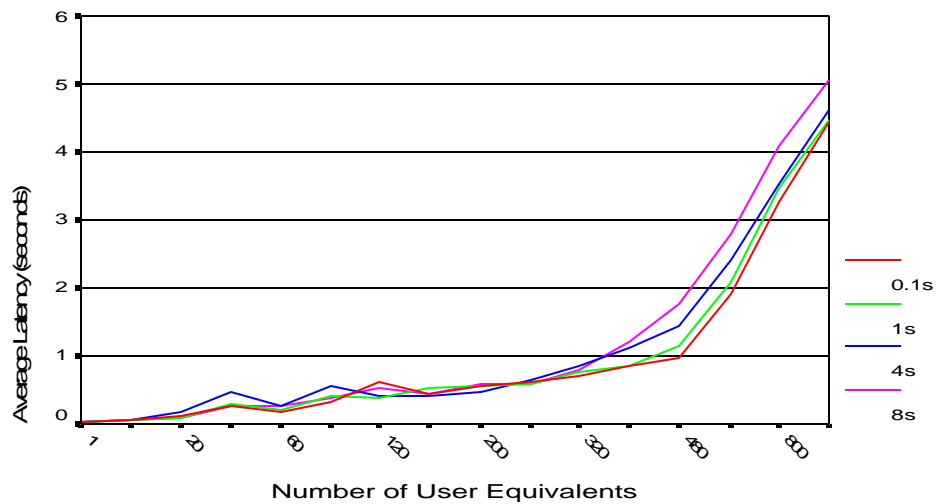


Figure 5.36 Comparison of average latencies with different averaging windows (RT-S, $s = 1.143$)

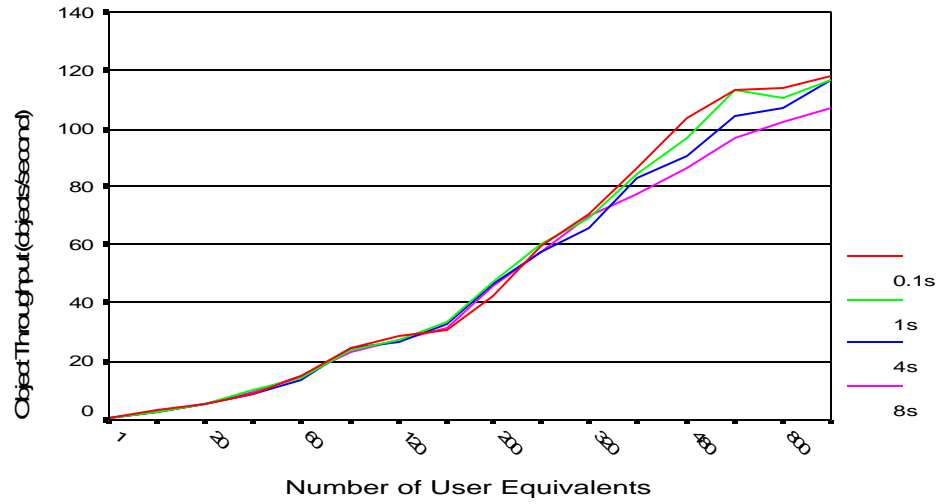


Figure 5.37 Comparison of object throughputs with different averaging windows (RT-G, $s = 1.143$)

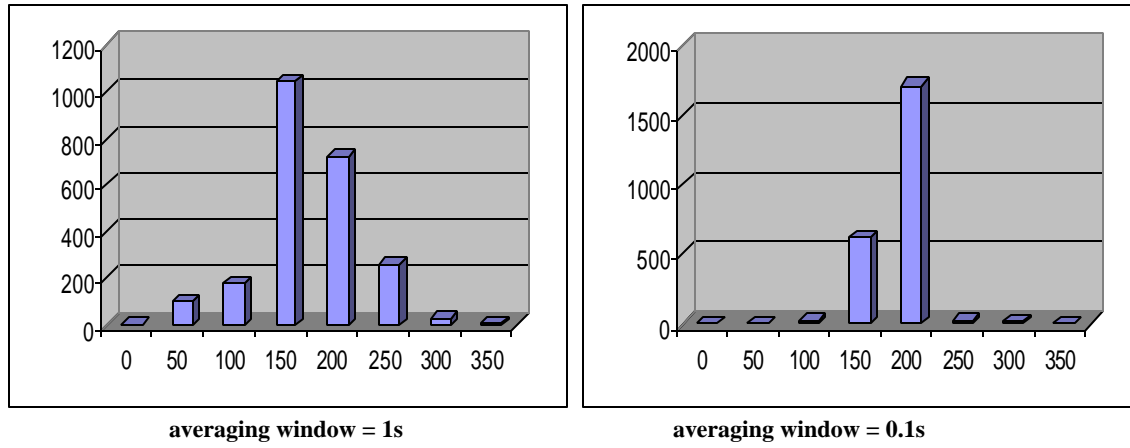


Figure 5.38 Rank of connections under RT-G policy $s = 1.143$, $N_{UE} = 640$

	mean	95% Confidence Interval	90-percentile
RR-G	2.113	(2.060, 2.167)	3.780
CONN-G	1.993	(1.943, 2.042)	2.591
RT-G-1s	2.323	(2.272, 2.374)	6.244
RT-G-0.1s	2.078	(2.022, 2.135)	2.719
RAN-G	2.149	(2.096, 2.201)	3.734

Table 5.6 Summary statistics ($s = 1.143$, $N_{UE} = 640$)

	Connections			Imbalance Index
	Mean	10 th Percentile	90 th Percentile	
RT-G-1s	154	96	216	1.39
RT-G-0.1s	155	144	163	1.06
CONN-G	149	136	162	1.09
RAN-G	152	110	196	1.28

Table 5.7 Summary statistics for connections RT-G, $s = 1.143$, $N_{UE} = 640$

5.3 Summary

In this chapter, we presented the structure of our simulation environment and ran a number of experiments on clustered server architectures. The results lead us to the following conclusions:

- “Switch” architectures were found to work better than “gateway” architectures, since the responses from servers are sent directly to users without going through the load balancer, which can become a bottleneck.
- The Least-Connection technique is found to be most efficient method in reducing average response time and load balancing. The average latencies for the Round-Robin and Random algorithms are slightly more than that for the Least-Connection policy for the types of traffic considered.
- The mean is sensitive to extreme outliers and this is not a good indication of low response times. A 90-percentile is more appropriate. The Least-Connection again achieves the best in terms of 90-percentile. We order the five algorithms from the slowest to the fastest when the averaging window is 1 second: XSIZE \rightarrow RT \rightarrow RAN \rightarrow RR \rightarrow CONN. A dynamic or run-time scheduling algorithm achieves the best performance. This result leads us to believe that it is sufficient to for load balancer to make their decisions solely on the basis of the connection state information.

- Simple load balancing algorithms, such as the Round-Robin, Random, and Least-Connection techniques, were found give better performance than the Round-Trip and Transmitted-Size methods (averaging window = 1s), which are both based on load statistics. The performance of algorithms that rely on statistical estimations is limited by the accuracy and variability of the collected data. Given small enough averaging windows, the Round-Trip algorithm can achieve the same performance level as the Least-Connection scheme. But it can only be implemented in the “Gateway” architecture.
- When the network is not a bottleneck, increasing the number of server stations in a cluster can significantly lower the average latency and increase throughput under heavy loads. The threshold of the maximum throughput increases with the increase of the servers in a cluster. An N -node cluster can achieve an average latency $1/N$ of that from a single server and a throughput approximately N times high than that of a single server.
- When the bandwidth of connection between the Internet and the cluster (load balancer in the case of “Gateway” architecture, load balancer and server stations in the case of “Switch” structure) creates a bottleneck, increasing it will substantially lower the average latency, and allow the clustered servers to achieve their maximum efficiency.
- Average file sizes have a negative effect on the response time and the load balancing. With the increase of S , fewer objects are served while more data are transferred. As to the load balancing, the “bytes transferred” measurement (percentage of bytes transferred) highly depends on the traffic pattern. The “objects served” measurement accurately reflects the server’s workload and is superior in representing load balancing.
- For those scheduling algorithms involving an averaging window, the clusters can achieve better results with small window size. The Least-Connection algorithms

with averaging window of 0.1s to 4s perform equally well. A threshold higher than 8s hurts performance. For the Round-Trip and Transmitted-Size schemes, an averaging window of 0.1s significantly boosts the performance.

Chapter 6

Conclusions

Web-server architectures that use clusters of workstations interconnected by a high-performance network are more scalable, cost-effective and reliable than single processor or tightly coupled multiprocessor systems. However, in implementing a clustered Web server it is crucial to use an effective load-balancing strategy that is matched to the traffic and user pattern seen by the server. Unfortunately, it is a difficult and involved task to evaluate the performance of alternative strategies. In this thesis, we presented a formula-based simulation system on clustering Web server technologies. Server models were developed for particular (but realistic!) traffic patterns and user behaviors by stress testing an Apache 1.3.12 web-server using the SURGE testing tool. In contrast to many simulation-based methods we were able to model the CPU time costs as both delays and as uses of resources: no connections are allowed once the CPU resource is exhausted, which prevents the response time from jumping dramatically towards infinity. Once the server models were developed and verified, it was straightforward to setup *ns2* to generate traffic that matches that used in the SURGE tests and to then experiment with different network and clustering configurations. This simulation system is used to demonstrate the feasibility of formula-based simulation in the study of load balancing and

to draw some conclusions regarding the performance of different load-balancing architectures and scheduling algorithms. We found no other studies on clustering servers solely based on a formula-based simulation system. In our literature review, we discovered some studies on real servers using real traffic traces [Bryhni 2000] [Carrera 2000]. Unfortunately, due to the complexity and extensive resources required, these studies could not be as extensive and controllable. Reconfiguring real servers is difficult and time-consuming.

Through the process we described in this thesis, we demonstrated that our approach is both valid and useful for choosing the best clustering architecture and scheduling algorithm suited for Web sites having specific content. After a straightforward hardware characterization via SURGE, the simulation system can be used to quickly survey a range of potential variations for a given distribution of server files and request parameters. Most important, the simulations predict expected behaviors of new protocols and designs without implementation costs or disruptions of existing system. The simulation system we built can be extended to include the characteristics of dynamic and secure requests and used to study the effect of these transactions on the clusters. SURGE programs could be modified to make dynamic or secure requests in order to come up with delay models for dynamic and secure requests. These models will then be integrated into our simulation system. Since the same traffic trace is used for SURGE and *ns2* simulations, our effort would be focused on building realistic server models.

Our models involved approximations to real world machines and networks as well as limitations in resources. The simulation system included all key system components (user, network, clustered server and load balancer models) and gave us valuable information about the general performance of the Web clustering technologies. To give us an idea of the accuracy of our simulations, we validated the server processing model

by comparing simulated results and measurements from SURGE experiments. The server processing models were shown to be reasonably close to the real world experience in terms of performance. At the beginning of our simulations, an alternative method from Bryhni was compared to our Least-Connection scheme to illustrate the difference between the HTTP/1.0 and HTTP/1.1-persistent connection protocols. The ability to replicate some of Bryhni's results is a strong indication that our simulation system works. Our work extends Bryhni's results to HTTP/1.1.

In the course of this thesis, we studied two clustering architectures, five scheduling algorithms and the effect that the number of servers in a cluster has on load balancing. Load balancers work either as gateways or switches and are responsible to choose the best server to satisfy incoming requests. Load balancing algorithms are meant to optimize request distributions according to Round-Robin, random schemes, current load (Least-Connection) or historical performance (Transmitted-Size, Round-Trip). Both raw performance data and load balancing data played an important role in this study. Our simulations showed that clustering servers is a good idea and that it reduces the response time and increases the throughput.

We drew some conclusions from our simulations:

- The 90-percentile is more appropriate in predicting the performance of response time. The measurement of means is highly sensitive to the extreme outliers.
- Clustered servers outperform traditional one-node servers. Given that the network does not create a bottleneck, Web clusters with N servers can reduce the average latency up to $1/N$ that of a traditional single server, and increase the throughput up to N times that of a single server.
- The replies should not be routed through the load balancer.

- For the traffic and server profiles studied, we find that load-balancing schemes that are based upon the number of active connections to each server give the best performance. Scheduling schemes such as Round-Robin and random are easy to implement and work generally as well or better than complicated schemes based on historical performance, such as Transmitted-Size and Round-Trip, in terms of reducing response time and better load balancing.
- The traffic pattern has a significant impact on the percentage of bytes transferred measurement. The percentage of objects transferred parameter is superior in representing the load distributions on the servers.

Some of the conclusions we drew from our experiments are similar to Bryhni's conclusions in his load balancing experiments in [Bryhni 2000]. For example, Round-Robin and Least-Connection schemes outperform scheduling algorithms based on history, such as Round-Trip and Transmitted-Size schemes. Also, the Round-Robin algorithm is recommended due to its simple implementation, low response time and good load balancing. We also studied the impact of varying the frequency of load information broadcasts. An averaging window ranging from 0.1 to 4 seconds performed equally well for Least-Connection policy while a threshold of 8 seconds deteriorates the performance. For Transmitted-Size and Round-Trip algorithms, the clusters can achieve better results in terms of response and throughput with small window size. Given small enough intervals between consecutive polls of the server state, the Round-Trip can achieve comparable performance levels to those of Least-Connection scheme.

Future studies need to be performed on the simulation system in order to improve the simulation system and have a better understanding of the load balancing systems. We suggest further work in the following five areas:

- 1) Current server processing models use the number of user equivalents N_{UE} associated with the server as a reference. We have three models under different average file sizes. They provide a rough estimation of the server delay under a certain average file sizes and N_{UE} . Future models could work on a smaller scale, attempting to give delays for each request that are a function of the number of concurrent open connections as well as size of the file being requested (perhaps with a random component to take into account other effects). Ideally the server model would be independent of the file-size distribution and the user parameter. This improved model would more precisely predict the server's workload.
- 2) The simulation system was built as a framework concentrating on the scheduling algorithms. The overheads incurred by IP-tunneling, IP modification or Network Address translation (NAT) were not fully taken into account by the simulation system. Further investigations are warranted to measure the impact of the overhead and delay characteristics of the load balancer.
- 3) We did not cover the cost involved with polling the servers for load information. A request/response process could be integrated into the simulation between the load balancer and the servers to inquiry about the servers' load, such as the number of connections. More experiments could be done to give us insights into how the frequency of polling affects the load balancer' performance in terms of delay.
- 4) We focused on static web-services in this study. In these services, requests are made only for HTML pages with embedded objects. There is a trend for dynamic and secure Web services. For dynamic Web services, objects are dynamically generated through the user and back-end server interactions. Typically, those services' requests are CPU bound and/or disk bound. Dynamic requests, as they are called, include all overheads of static requests as well as overheads due to back-end server computations needed to generate dynamic objects. Servers with

CGI scripts will need extra computing resources in order to work. Every time a script is invoked, the HTTP server starts another program to execute the script. Depending on the task performed by the script, it may require significant processor time and memory. For secure Web services, dynamic pages are requested over a secure connection. These requests tend to be CPU bound because of the overheads necessary to setup a secure connection and to execute cryptographic algorithms. Cardellini [2001] showed that the operations of encryption and decryption are very critical tasks. A very limited increase in client arrivals is sufficient to congest CPU system's queues. It would be interesting to create server models for dynamic and secure Web services. Some modifications on SURGE would allow requests for dynamic and secure requests and would retain SURGE's probabilistic models and flexibility. Research on the user's behaviors for dynamic and secure pages is required in order for SURGE to have a realistic traffic pattern.

- 5) The Master/Slave architecture mentioned in Chapter 2 is directly related to the dynamic or secure Web services. Servers executing many CGI scripts will have a lot of overhead as well as many processes competing for the processor and memory of the server. Using a CGI script to dynamically generate a Web document is therefore usually much slower than directly retrieving a file of the same size. The Master/Slave approach separates dynamic and static content processing, so long running, resource intensive CGI scripts will not slow down simple, static content requests. We would like to see how efficient this Master/slave architecture is in reducing response time.

Appendix A:

Simulation

Simulation packages:

“Four widely simulators today include OPNET [OPNET Technologies 2002], *ns2* [Fall], PARSEC [Laboratory] and SSF [Network]. At a high level all of these simulators are similar: they focus on packet-level discrete-event simulation and they model a wide range of protocols in the traditional Internet protocol suite. Each simulator has a slightly different focus. OPNET is a commercial simulator with strong customer support, while the others are targeted primarily at the protocol design and networking communities. *ns2* emphasizes support for a wide range of wired and wireless protocols and multiple levels of abstraction. PARSEC provides high performance parallel simulation and focuses primarily on wireless simulation. SSF also provides parallelism, but it emphasizes support for routing protocol simulation and for very large scale.” [Heidemann 2001]

***ns2* Simulator:**

In this project, *ns2* was chosen as the simulator. It provides the simulation programmer with an easy-to-use, reconfigurable, programmable simulation environment. “The Virtual InterNetwork Testbed (VINT) project has developed *ns2* as a common simulator with advanced feature to change current protocol engineering practices by enabling the study of protocol interactions and scaling.

ns2 provides several levels of abstraction:

- The default simulator provides a detailed model with hop-by-hop packet forwarding and dynamic routing updates.
- Centralized routing replaces routing messages with a centralized computation, saving processing time and memory in exchange for slightly different timing in routing changes.
- Session-level packet forwarding replaces hop-by-hop packet flow with a precomputed propagation delay.
- Algorithmic routing replaces shortest-path routing with tree-based routing, transforming $O(n \log n)$ memory requirements to $O(n)$.” [Breslau 2000]

ns2 provides a split-level programming model in which packet processing is done in a systems language while simulation setup is done in a scripting language. The split-programming model provides adequate flexibility without unduly constraining performance. In particular, tasks such as low-level event processing or packet forwarding through a simulated router require high performance and are modified infrequently once put into place. Thus, they are best served by expressing an implementation in a compiled language such as C++. On the other hand, tasks such as the dynamic configuration of protocol objects and the specification and placement of traffic sources are often iteratively refined and undergo frequent change as the research task unfolds. Thus, they are best served by an implantation in a flexible and interactive scripting language such as Tcl.

Appendix B:

SURGE Measurements

1	1.991	5	1.444	9	1.291	13	2.2
2	1.493	6	1.304	10	1.428	14	1.563
3	1.283	7	1.867	11	1.402	15	2.097
4	1.532	8	1.288	12	1.541	average	1.5816

Figure B.1 Roundtrip time measured by “traceroute” command

1	851.71	6	832.77	11	836.07	16	845.46
2	831.38	7	831.45	12	837.45	17	840.66
3	819.26	8	850.41	13	858.38	18	861.74
4	816.11	9	843.97	14	824.92	average	841.8133
5	847.19	10	848.40	15	875.31	variance	227.1606

Figure B.2 Bandwidth measured by FTP requests

Number of UE's	S = 1.143		S = 1.7145		S = 2.5	
	I	II	I	II	I	II
1	.015	.016	.020	.020	.047	.058
10	.026	.028	.031	.032	.059	.059
20	.035	.034	.036	.038	.076	.075
40	.049	.056	.069	.082	.093	.143
60	.071	.066	.101	.105	.177	.174
100	.223	.183	.368	.344	.791	.796
120	.843	.353	1.055	.563	1.534	1.306
140	1.170	.598	1.505	.998	2.348	2.112
200	3.665	1.892	3.995	2.327	4.672	3.402
260	5.778	3.420	6.477	4.050	6.948	5.066
320	8.036	4.907	8.483	5.637	8.966	6.922
400	11.039	6.735	11.159	7.428	11.812	9.789
480	13.075	8.082	13.453	9.213	13.682	12.062
640	18.543	11.957	18.546	12.181	19.327	15.402
800	22.116	14.956	22.751	16.082	22.549	19.350
960	25.589	17.268	26.464	18.912	26.495	21.546

Table B.3 Measured latencies of Web objects from two Apache configurations

Number of UE's	S = 1.143				
	I	II	III	Average latency	Standard Deviation
1	0.017	0.016	0.016	0.016	0.001
10	0.030	0.026	0.028	0.028	0.002
20	0.035	0.033	0.033	0.034	0.001
40	0.053	0.058	0.055	0.056	0.003
60	0.070	0.067	0.061	0.066	0.005
100	0.196	0.172	0.181	0.183	0.012
120	0.341	0.331	0.386	0.353	0.029
140	0.600	0.550	0.645	0.598	0.048
200	1.780	1.869	2.028	1.892	0.126
260	3.203	3.505	3.553	3.420	0.190
320	4.648	5.150	4.923	4.907	0.251
400	6.365	6.996	6.843	6.735	0.329
480	7.665	8.367	8.212	8.082	0.369
640	11.016	12.170	12.686	11.957	0.855
800	14.573	15.875	14.419	14.956	0.800
960	17.247	16.524	18.031	17.268	0.754

Figure B.4 Average latencies for S=1.143

Number of UE's	s = 1.143				
	I	II	III	Average Bytes /second	Standard Deviation
1	2867.1	2867.1	2867.1	2867.1	0.000
10	49012.5	51517.6	51517.6	50682.6	1446.320
20	123298.8	123343.8	123295.3	123312.6	27.048
40	242528.1	241935.9	242655.5	242373.1	384.004
60	357090.0	357111.6	356672.1	356957.9	247.746
100	583958.7	587516.1	587856.9	586443.9	2158.981
120	683928.2	668296.7	671812.7	674679.2	8200.522
140	711890.8	705191.9	697694.2	704925.6	7102.045
200	750135.8	738088.2	747383.6	745202.5	6312.999
260	762221.3	756028.4	719730.2	745993.3	22954.320
320	734229.0	735779.2	704343.0	724783.7	17719.155
400	710467.4	709448.8	693105.6	704340.6	9743.116
480	690996.0	687051.5	687117.9	688388.5	2258.434
640	589108.9	610778.9	625299.6	608395.8	18212.662
800	614251.0	562634.7	623759.7	600215.1	32891.041
960	581205.0	611325.3	533127.4	575219.2	39441.095

Figure B.5 Byte throughputs for $s=1.143$

Number of UE's	s = 1.143				
	I	II	III	Average HTTP Ops/second	Standard Deviation
1	0.37	0.37	0.37	0.37	0.000
10	4.05	4.05	4.05	4.05	0.000
20	9.37	9.37	9.37	9.37	0.000
40	18.99	18.94	19.00	18.98	0.032
60	24.61	24.62	24.58	24.60	0.021
100	41.17	41.38	41.43	41.32	0.138
120	47.68	46.88	46.79	47.11	0.490
140	48.74	48.17	47.68	48.19	0.531
200	51.29	50.59	51.07	50.98	0.358
260	52.03	51.51	49.24	50.93	1.484
320	50.29	50.49	48.06	49.61	1.349
400	48.35	48.17	47.24	47.92	0.596
480	46.97	46.70	46.29	46.65	0.342
640	39.89	41.42	42.28	41.20	1.211
800	41.36	37.97	41.96	40.43	2.151
960	38.82	41.07	36.29	38.73	2.391

Figure B.6 HTTP ops/second for $s=1.143$

Number of UE's	S = 1.143				
	I	II	III	Average objects /second	Standard deviation
1	0.29	0.29	0.29	0.29	0.000
10	2.75	2.76	2.75	2.75	0.006
20	5.89	5.90	5.90	5.90	0.006
40	11.86	11.84	11.82	11.84	0.020
60	17.71	17.34	17.79	17.62	0.240
100	28.60	28.49	28.47	28.52	0.070
120	33.00	30.42	33.45	32.29	1.635
140	35.94	35.55	34.98	35.49	0.483
200	38.43	37.61	36.53	37.52	0.953
260	38.19	37.27	36.11	37.19	1.042
320	36.88	36.71	35.60	36.40	0.695
400	35.40	35.88	35.92	35.73	0.289
480	34.45	35.22	33.79	34.49	0.716
640	33.05	33.74	32.55	33.11	0.598
800	33.42	30.89	33.28	32.53	1.422
960	32.13	29.72	28.73	30.19	1.749

Figure B.7 Object throughputs for $s=1.143$

Number of UE's	S = 1.143				
	I	II	III	Average number of timeout's	Standard deviation
320	0	2	0	1	1.155
400	3	8	5	5	2.517
480	49	50	42	47	4.359
640	237	247	235	240	6.429
800	455	499	433	462	33.606
960	722	791	770	761	35.369

Figure B.8 Numbers of timeouts for $s=1.143$

Number of UE's	$s = 1.7145$				
	I	II	III	Average latency	Standard Deviation
1	0.020	0.020	0.020	0.020	0.000
10	0.031	0.032	0.032	0.032	0.001
20	0.038	0.036	0.040	0.038	0.002
40	0.081	0.077	0.089	0.082	0.006
60	0.105	0.108	0.104	0.105	0.002
100	0.338	0.348	0.346	0.344	0.005
120	0.562	0.582	0.544	0.563	0.019
140	1.024	0.972	0.999	0.998	0.026
200	2.287	2.410	2.282	2.327	0.073
260	3.870	4.023	4.258	4.050	0.195
320	5.477	5.757	5.678	5.637	0.144
400	7.558	7.559	7.166	7.428	0.227
480	9.361	9.203	9.076	9.213	0.143
640	12.257	12.134	12.153	12.181	0.066
800	16.714	15.431	16.101	16.082	0.642
960	19.243	19.418	18.076	18.912	0.730

Figure B.9 Average latencies for $s=1.7145$

Number of UE's	$s = 1.7145$				
	I	II	III	Average Bytes /second	Standard deviation
1	4019.1	4019.1	4019.1	4019.1	0.000
10	65370.5	65370.5	65370.5	65371.4	0.000
20	147760.6	147738.3	147752.8	147750.6	11.317
40	333851.8	333051.2	333050.1	333317.7	462.544
60	423552.8	424208.6	422746.2	423502.5	732.495
100	656563.6	659146.1	652723.9	656144.6	3231.544
120	718507.7	730589.0	716724.6	721940.4	7542.754
140	755186.7	758807.3	747676.2	753890.1	5677.701
200	774949.5	780370.1	779689.0	778336.2	2952.673
260	785937.6	781215.1	784303.1	783818.6	2398.240
320	777202.1	759435.1	777627.2	771421.5	10382.674
400	748628.3	731413.2	749876.8	743306.1	10318.454
480	730048.9	707419.6	742122.8	726530.4	17617.115
640	686209.9	690268.5	721416.7	699298.4	19262.231
800	682232.6	676041.4	663457.2	673910.4	9567.381
960	656865.4	674449.7	650383.0	660566.0	12452.812

Figure B.10 Byte throughputs for $s=1.7145$

Number of UE's				$s = 1.7145$	Average HTTP ops/second	Standard deviation
	I	II	III			
1	0.36	0.36	0.36		0.36	0.000
10	3.72	3.73	3.73		3.73	0.006
20	8.35	8.34	8.38		8.36	0.021
40	16.91	16.95	17.05		16.97	0.072
60	22.74	22.78	22.75		22.76	0.021
100	31.78	31.35	31.64		31.59	0.219
120	33.25	33.07	33.31		33.21	0.125
140	29.87	30.20	29.75		29.94	0.233
200	31.69	31.70	32.46		31.95	0.442
260	32.48	32.74	33.36		32.86	0.452
320	32.69	31.67	33.44		32.60	0.888
400	31.21	31.55	31.62		31.46	0.219
480	30.40	30.72	31.36		30.82	0.489
640	30.01	30.89	30.73		30.54	0.469
800	29.46	30.13	29.44		29.68	0.393
960	29.55	29.18	29.33		29.35	0.186

Figure B.11 HTTP ops/second for $s=1.7145$

Number of UE's				$s = 1.7145$	Average objects /second	Standard deviation
	I	II	III			
1	0.29	0.29	0.29		0.29	0.000
10	2.75	2.75	2.74		2.75	0.006
20	5.89	5.88	5.88		5.88	0.006
40	11.66	11.65	11.67		11.66	0.010
60	17.45	17.43	17.25		17.38	0.110
100	26.74	26.02	26.16		26.31	0.382
120	28.57	28.00	27.59		28.05	0.492
140	30.75	32.46	31.83		31.68	0.865
200	32.30	34.00	34.29		33.53	1.075
260	33.01	31.01	34.21		32.74	1.617
320	31.82	31.85	33.13		32.27	0.748
400	30.76	30.65	32.37		31.26	0.963
480	30.26	29.58	31.57		30.47	1.011
640	28.72	28.04	30.56		29.11	1.304
800	26.69	28.80	30.43		28.64	1.875
960	25.33	27.68	29.22		27.41	1.959

Figure B.12 Object throughputs for $s=1.7145$

Number of UE's				s = 1.7145	
	I	II	III	Average number of timeout's	Standard deviation
320	2	0	0	1	1.155
400	2	5	9	5	3.512
480	67	56	54	59	7.000
640	247	240	235	241	6.028
800	512	488	455	485	28.618
960	825	773	738	779	43.776

Figure B.13 Numbers of timeouts for $s=1.7145$

Number of UE's				s = 2.5	
	I	II	III	Average latency	Standard deviation
1	0.078	0.044	0.052	0.058	0.018
10	0.059	0.059	0.061	0.059	0.001
20	0.085	0.072	0.067	0.075	0.009
40	0.188	0.122	0.118	0.143	0.039
60	0.179	0.171	0.172	0.174	0.004
100	0.770	0.802	0.817	0.796	0.024
120	1.261	1.308	1.351	1.306	0.045
140	2.237	2.132	1.966	2.112	0.137
200	3.692	3.245	3.270	3.402	0.251
260	5.350	4.803	5.046	5.066	0.274
320	7.565	6.557	6.645	6.922	0.558
400	10.279	9.597	9.491	9.789	0.428
480	12.295	12.269	11.622	12.062	0.381
640	15.900	15.047	15.258	15.402	0.444
800	19.928	18.821	19.301	19.350	0.555
960	22.739	21.468	20.432	21.546	1.155

Figure B.14 Average latencies for $s=2.5$

Number of UE's	s = 2.5			Average Bytes /second	Standard deviation
	I	II	III		
1	12040.3	12040.3	12040.3	12040.3	12040.3
10	122714.8	122789.7	122850.7	122785.1	122714.8
20	226652.9	226464.7	226919.9	226679.2	226652.9
40	407295.4	407882.8	409187.9	408122.0	407295.4
60	521881.2	521873.3	522220.7	521991.7	521881.2
100	742316.1	732661.0	738909.1	737962.1	742316.1
120	771984.9	765939.0	773311.5	770411.8	771984.9
140	769722.4	784327.4	792647.4	782232.4	769722.4
200	759349.1	780686.6	767975.5	769337.1	759349.1
260	756654.9	757412.7	770107.9	761391.8	756654.9
320	768497.4	747840.3	780494.2	765610.6	768497.4
400	772455.4	748934.6	790141.9	770510.6	772455.4
480	765541.1	770968.4	783290.8	773266.7	765541.1
640	740898.4	785418.9	775920.2	767412.5	740898.4
800	757693.2	756518.4	771035.5	761749.0	757693.2
960	757642.8	769077.3	755014.7	760578.3	757642.8

Figure B.15 Byte throughputs for $s=2.5$

Number of UE's	s = 2.5			Average objects /second	Standard deviation
	I	II	III		
1	0.29	0.29	0.29	0.29	0.000
10	2.71	2.71	2.71	2.71	0.000
20	5.83	5.83	5.82	5.82	0.006
40	11.44	11.46	11.46	11.45	0.012
60	17.09	17.14	16.95	17.06	0.098
100	23.56	23.84	23.35	23.58	0.246
120	24.64	24.68	23.68	24.33	0.566
140	23.81	23.78	24.51	24.03	0.413
200	25.35	24.49	26.78	25.54	1.157
260	25.65	25.10	26.70	25.81	0.813
320	25.58	25.45	26.28	25.77	0.446
400	25.58	25.36	26.20	25.72	0.436
480	25.32	25.79	25.79	25.63	0.271
640	24.50	25.03	24.95	24.82	0.286
800	24.82	24.59	24.24	24.55	0.292
960	24.87	24.46	23.67	24.33	0.610

Figure B.16 Object throughputs for $s=2.5$

Number of UE's				S = 2.5	
	I	II	III	Average HTTP ops/second	Standard deviation
1	0.36	0.36	0.36	0.36	0.000
10	3.72	3.73	3.73	3.73	0.006
20	8.35	8.34	8.38	8.36	0.021
40	16.91	16.95	17.05	16.97	0.072
60	22.74	22.78	22.75	22.76	0.021
100	31.78	31.35	31.64	31.59	0.219
120	33.25	33.07	33.31	33.21	0.125
140	29.87	30.20	29.75	29.94	0.233
200	31.69	31.70	32.46	31.95	0.442
260	32.48	32.74	33.36	32.86	0.452
320	32.69	31.67	33.44	32.60	0.888
400	31.21	31.55	31.62	31.46	0.219
480	30.40	30.72	31.36	30.82	0.489
640	30.01	30.89	30.73	30.54	0.469
800	29.46	30.13	29.44	29.68	0.393
960	29.55	29.18	29.33	29.35	0.186

Figure B.17 HTTP ops/second for $s=2.5$

Number of UE's				S = 2.5	
	I	II	III	Average number of timeout's	Standard deviation
320	0	0	0	0	0.000
400	1	2	0	1	1.000
480	30	37	30	32	4.041
640	236	216	231	228	10.408
800	505	456	557	506	50.507
960	750	739	770	753	15.716

Figure B.18 Numbers of timeouts for $s=2.5$

Number of UE's		$s = 1.143$			
	Bytes /second	HTTP ops /second	Objects /second	Number of Timeouts	Percentage of Timeouts
1	2867.1	0.37	0.29	0	0
10	50682.6	4.05	2.75	0	0
20	123312.6	9.37	5.90	0	0
40	242373.1	18.98	11.84	0	0
60	356957.9	24.60	17.62	0	0
100	586443.9	41.32	28.52	0	0
120	674679.2	47.11	32.29	0	0
140	704925.6	48.19	35.49	0	0
200	745202.5	50.98	37.52	0	0
260	745993.3	50.93	37.19	0	0
320	724783.7	49.61	36.40	1	0.00%
400	704340.6	47.92	35.73	5	0.02%
480	688388.5	46.65	34.49	47	0.23%
640	608395.8	41.20	33.11	240	1.21%
800	600215.1	40.43	32.53	462	2.37%
960	575219.2	38.73	30.19	761	4.20%

Table B.19 Throughput and the number of requests rejected ($s=1.143$)

Number of UE's		$s = 1.7145$			
	Bytes /second	HTTP ops /second	Objects /second	Number of Timeouts	Percentage of Timeouts
1	4019.1	0.37	0.29	0	0
10	65371.4	4.07	2.75	0	0
20	147750.6	9.13	5.88	0	0
40	333317.7	18.70	11.66	0	0
60	423502.5	24.34	17.38	0	0
100	656144.6	39.09	26.31	0	0
120	721940.4	43.47	28.05	0	0
140	753890.1	42.15	31.68	0	0
200	778336.2	42.80	33.53	0	0
260	783818.6	43.45	32.74	0	0
320	771421.5	42.49	32.27	1	0.00%
400	743306.1	40.92	31.26	5	0.03%
480	726530.4	40.22	30.47	59	0.32%
640	699298.4	38.08	29.11	241	1.38%
800	673910.4	36.63	28.64	485	2.82%
960	660566.0	35.96	27.41	779	4.73%

Table B.20 Throughput and the number of requests rejected ($s=1.7145$)

Number of UE's	$s = 2.5$				
	Bytes /second	HTTP ops /second	Objects /second	Number of Timeouts	Percentage of Timeouts
1	12040.3	0.36	0.29	0	0
10	122785.1	3.73	2.71	0	0
20	226679.2	8.36	5.82	0	0
40	408122.0	16.97	11.45	0	0
60	521991.7	22.76	17.06	0	0
100	737962.1	31.59	23.58	0	0
120	770411.8	33.21	24.33	0	0
140	782232.4	29.94	24.03	0	0
200	769337.1	31.95	25.54	0	0
260	761391.8	32.86	25.81	0	0
320	765610.6	32.60	25.77	0	0.00%
400	770510.6	31.46	25.72	1	0.01%
480	773266.7	30.82	25.63	32	0.21%
640	767412.5	30.54	24.82	228	1.53%
800	761749.0	29.68	24.55	506	3.44%
960	760578.3	29.35	24.33	753	5.16%

Table B.21 Throughput and the number of requests rejected ($s=2.5$)

Appendix C:

Server Models (Integration and Validation)

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	0.035	0.035	0.040	0.037	0.016	0.436
10	0.069	0.069	0.070	0.069	0.028	0.404
20	0.094	0.094	0.090	0.093	0.034	0.367
40	0.152	0.200	0.140	0.164	0.056	0.341
60	0.172	0.175	0.170	0.172	0.066	0.383
100	0.254	0.283	0.260	0.266	0.183	0.689
120	0.365	0.368	0.370	0.368	0.353	0.960
140	0.537	0.550	0.550	0.546	0.598	1.096
200	2.244	2.283	2.240	2.256	1.892	0.839
260	3.962	4.001	4.010	3.991	3.420	0.857
320	5.727	5.669	5.670	5.689	4.907	0.863
400	7.912	7.963	7.930	7.935	6.735	0.849
480	10.168	10.151	10.210	10.176	8.082	0.794
640	14.557	14.529	14.530	14.539	11.957	0.822
800	18.813	18.829	18.800	18.814	14.956	0.795
960	22.959	22.900	22.950	22.936	17.268	0.753

Table C.1 Average latencies, $s = 1.143$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	0.288	0.288	0.290	0.289	0.290	1.004
10	2.717	2.717	2.720	2.718	2.750	1.012
20	5.803	5.803	5.800	5.802	5.900	1.017
40	11.147	10.735	10.950	10.944	11.840	1.082
60	17.232	17.210	17.270	17.237	17.620	1.022
100	28.320	28.098	28.250	28.223	28.520	1.011
120	33.190	33.120	33.170	33.160	32.290	0.974
140	36.940	36.832	36.720	36.831	35.490	0.964
200	36.388	36.148	36.430	36.322	37.520	1.033
260	35.962	35.775	35.720	35.819	37.190	1.038
320	35.460	35.690	35.690	35.613	36.400	1.022
400	35.435	35.345	35.460	35.413	35.730	1.009
480	34.697	34.342	34.680	34.573	34.490	0.998
640	33.560	33.353	33.410	33.441	33.110	0.990
800	31.847	31.882	31.880	31.869	32.530	1.021
960	30.750	30.502	30.650	30.634	30.190	0.986

Table C.2 Object throughputs, $s = 1.143$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	2867.1	2867.1	2867.1	2867.1	2867.1	1.000
10	47491.3	47491.3	47491.3	47491.3	50682.6	1.067
20	121496.5	121496.5	121496.5	121496.5	123312.6	1.015
40	229902.6	218014.4	225488.4	224468.5	242373.1	1.080
60	353500.7	352680.8	354062.0	353414.5	356957.9	1.010
100	581970.8	578636.9	580762.0	580456.6	586443.9	1.010
120	681105.5	680219.9	680979.8	680768.4	674679.2	0.991
140	746742.9	744850.4	742708.9	744767.4	704925.6	0.947
200	737877.1	732861.2	739668.5	736802.2	745202.5	1.011
260	730419.0	726288.8	725421.1	727376.3	745993.3	1.026
320	716552.9	721498.5	721557.5	719869.6	724783.7	1.007
400	716302.5	714710.6	716484.1	715832.4	704340.6	0.984
480	702179.8	696773.8	702185.0	700379.5	688388.5	0.983
640	681093.9	676924.6	677347.5	678455.3	608395.8	0.897
800	644519.0	646730.9	647201.3	646150.4	600215.1	0.929
960	621901.5	618569.6	620975.4	620482.2	575219.2	0.927

Table C.3 Byte throughputs, $s = 1.143$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	0.051	0.051	0.050	0.051	0.020	0.395
10	0.074	0.074	0.070	0.073	0.032	0.440
20	0.104	0.104	0.100	0.103	0.038	0.370
40	0.173	0.239	0.240	0.217	0.082	0.377
60	0.209	0.230	0.290	0.243	0.105	0.432
100	0.414	0.422	0.420	0.419	0.344	0.822
120	0.605	0.588	0.590	0.594	0.563	0.947
140	0.831	0.871	0.870	0.857	0.998	1.164
200	2.900	2.902	2.860	2.887	2.327	0.806
260	4.676	4.685	4.650	4.670	4.050	0.867
320	6.400	6.495	6.490	6.462	5.637	0.872
400	8.818	8.820	8.850	8.829	7.428	0.841
480	11.110	11.151	11.130	11.130	9.213	0.828
640	15.562	15.552	15.540	15.551	12.181	0.783
800	19.769	19.814	19.740	19.774	16.082	0.813
960	23.858	23.887	23.810	23.852	18.912	0.793

Table C.4 Average latencies, $s = 1.7145$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	0.288	0.288	0.290	0.289	0.290	1.004
10	2.715	2.715	2.710	2.713	2.750	1.014
20	5.785	5.785	5.790	5.787	5.880	1.016
40	11.297	11.175	11.180	11.217	11.660	1.039
60	16.985	16.888	16.640	16.838	17.380	1.032
100	27.107	27.042	27.080	27.076	26.310	0.972
120	31.133	31.265	31.260	31.219	28.050	0.898
140	34.158	33.802	33.840	33.933	31.680	0.934
200	32.378	32.322	32.590	32.430	33.530	1.034
260	32.633	32.582	32.710	32.642	32.740	1.003
320	32.943	32.597	32.630	32.723	32.270	0.986
400	32.860	32.843	32.800	32.834	31.260	0.952
480	32.595	32.330	32.430	32.452	30.470	0.939
640	31.127	31.020	30.910	31.019	29.110	0.938
800	29.843	29.480	29.550	29.624	28.640	0.967
960	28.953	29.103	29.090	29.049	27.410	0.944

Table C.5 Object throughputs, $s = 1.7145$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	4019.1	4019.1	4019.1	4019.1	4019.1	1.000
10	63144.7	63144.7	65371.4	63886.9	65371.4	1.023
20	145411.5	145411.5	147750.6	146191.2	147750.6	1.011
40	326628.9	321852.7	333317.7	327266.4	333317.7	1.018
60	429782.2	428241.6	423502.5	427175.4	423502.5	0.991
100	708944.0	708056.3	656144.6	691048.3	656144.6	0.949
120	810475.9	813083.3	721940.4	781833.2	721940.4	0.923
140	850518.9	842080.6	753890.1	815496.6	753890.1	0.924
200	811985.3	810907.1	778336.2	800409.5	778336.2	0.972
260	816807.9	815668.4	783818.6	805431.7	783818.6	0.973
320	824352.9	817393.6	771421.5	804389.4	771421.5	0.959
400	821974.4	821766.5	743306.1	795682.3	743306.1	0.934
480	815892.7	808987.1	726530.4	783803.4	726530.4	0.927
640	781206.9	778726.4	699298.4	753077.2	699298.4	0.929
800	744056.4	735075.4	673910.4	717680.7	673910.4	0.939
960	723811.4	726587.7	660566.0	703655.1	660566.0	0.939

Table C.6 Byte throughputs, $s = 1.7145$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	0.133	0.133	0.130	0.132	0.058	0.439
10	0.178	0.178	0.180	0.179	0.059	0.330
20	0.189	0.189	0.190	0.189	0.075	0.396
40	0.231	0.239	0.230	0.233	0.143	0.613
60	0.315	0.319	0.300	0.311	0.174	0.559
100	0.795	0.775	0.810	0.793	0.796	1.003
120	1.221	1.243	1.220	1.228	1.306	1.064
140	1.993	2.001	1.960	1.985	2.112	1.064
200	4.311	4.336	4.390	4.346	3.402	0.783
260	6.326	6.295	6.300	6.307	5.066	0.803
320	8.296	8.292	8.330	8.306	6.922	0.833
400	10.949	10.915	10.890	10.918	9.789	0.897
480	13.382	13.384	13.390	13.385	12.062	0.901
640	18.012	18.057	17.970	18.013	15.402	0.855
800	22.428	22.375	22.410	22.404	19.350	0.864
960	26.463	26.452	26.440	26.452	21.546	0.815

Table C.7 Average latencies, $s = 2.5$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	0.288	0.288	0.290	0.289	0.290	1.004
10	2.150	2.150	2.150	2.150	2.710	1.260
20	5.643	5.643	5.640	5.642	5.820	1.032
40	11.208	11.180	11.210	11.199	11.450	1.022
60	16.387	16.508	16.580	16.492	17.060	1.034
100	24.570	24.710	24.520	24.600	23.580	0.959
120	26.668	26.567	26.670	26.635	24.330	0.913
140	26.492	26.457	26.680	26.543	24.030	0.905
200	26.222	26.122	25.950	26.098	25.540	0.979
260	26.892	26.953	26.970	26.938	25.810	0.958
320	27.425	27.438	27.360	27.408	25.770	0.940
400	27.823	27.897	27.930	27.883	25.720	0.922
480	27.875	27.875	27.870	27.873	25.630	0.920
640	27.345	27.430	27.420	27.398	24.820	0.906
800	26.723	26.407	26.750	26.627	24.550	0.922
960	26.087	25.993	26.110	26.063	24.330	0.933

Table C.8 Object throughputs, $s = 2.5$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	SURGE	Ratio (Surge/Mean)
1	12040.3	12040.3	12040.3	12040.3	12040.3	1.000
10	104842.1	104842.1	104842.1	104842.1	122785.1	1.171
20	223568.7	223568.7	223568.7	223568.7	226679.2	1.014
40	401877.2	401390.9	401877.2	401715.1	408122.0	1.016
60	564589.2	567425.7	568840.8	566951.9	521991.7	0.921
100	843443.8	846490.2	842302.8	844079.0	737962.1	0.874
120	903524.3	892518.1	898300.4	898114.3	770411.8	0.858
140	888010.5	887198.1	891502.0	888903.5	782232.4	0.880
200	845262.6	843362.3	826652.0	838425.6	769337.1	0.918
260	874003.6	880128.4	880380.9	878170.9	761391.8	0.867
320	914534.2	914883.3	912883.6	914100.4	765610.6	0.838
400	924483.0	925867.3	926329.7	925560.0	770510.6	0.832
480	925738.4	925459.0	925188.0	925461.8	773266.7	0.836
640	910505.1	914502.3	912403.5	912470.3	767412.5	0.841
800	893054.1	885244.8	893994.9	890764.6	761749.0	0.855
960	876882.5	873108.9	877450.9	875814.1	760578.3	0.868

Table C.9 Byte throughputs, $s = 2.5$, 10 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	0.027	0.027	0.027	0.027	0.027	1.000
10	0.062	0.062	0.062	0.062	0.062	1.000
20	0.094	0.091	0.094	0.093	0.093	1.004
40	0.177	0.249	0.213	0.213	0.225	0.947
60	0.370	0.303	0.303	0.325	0.310	1.048
100	0.766	0.437	0.568	0.590	0.532	1.110
120	0.682	0.637	0.868	0.729	0.745	0.979
140	0.865	0.797	0.848	0.837	0.827	1.011
200	2.588	2.833	2.701	2.707	2.747	0.986
260	4.527	4.374	4.302	4.401	4.359	1.010
320	5.984	6.039	6.053	6.025	6.039	0.998
400	8.250	8.325	8.312	8.296	8.311	0.998
480	10.368	10.566	10.319	10.418	10.434	0.998
640	14.775	14.794	14.736	14.768	14.766	1.000
800	18.909	18.904	18.926	18.913	18.914	1.000
960	23.161	23.026	23.107	23.098	23.077	1.001

Table C.10 Average latencies, $s = 1.143$, 100 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	0.288	0.288	0.288	0.288	0.288	1.000
10	2.722	2.722	2.722	2.722	2.722	1.000
20	4.655	4.807	4.655	4.706	4.722	0.996
40	9.377	10.188	9.813	9.793	9.931	0.986
60	15.312	16.193	14.235	15.247	15.225	1.001
100	22.290	21.800	22.852	22.314	22.322	1.000
120	26.412	26.062	25.043	25.839	25.648	1.007
140	30.308	30.282	31.292	30.627	30.734	0.997
200	30.420	28.855	31.065	30.113	30.011	1.003
260	31.718	31.267	32.262	31.749	31.759	1.000
320	33.083	32.552	32.458	32.698	32.569	1.004
400	33.253	32.160	32.375	32.596	32.377	1.007
480	33.037	32.748	33.412	33.066	33.075	1.000
640	32.058	32.120	32.033	32.071	32.075	1.000
800	30.517	30.948	31.227	30.897	31.024	0.996
960	29.815	30.223	30.148	30.062	30.145	0.997

Table C.11 Object throughputs, $s = 1.143$, 100 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	2867.1	2867.1	2867.1	2867.1	2867.1	1.000
10	47607.3	47607.3	47607.3	47607.3	47607.3	1.000
20	97705.8	100808.6	97705.8	98740.1	99084.9	0.997
40	192067.9	206576.8	199761.3	199468.7	201935.6	0.988
60	315918.4	333030.4	294344.5	314431.1	313935.3	1.002
100	459257.5	446787.9	471328.5	459124.6	459080.3	1.000
120	544019.0	538149.5	519949.0	534039.2	530712.6	1.006
140	613210.0	613149.5	631092.3	619150.6	621130.8	0.997
200	611463.4	582213.1	624946.8	606207.8	604455.9	1.003
260	638381.5	629767.7	648074.8	638741.4	638861.3	1.000
320	673244.9	659989.0	660888.4	664707.4	661861.6	1.004
400	674189.3	653021.5	658177.3	661796.0	657664.9	1.006
480	670655.9	665584.3	679670.5	671970.2	672408.3	0.999
640	652440.7	654546.4	653144.0	653377.0	653689.1	1.000
800	619681.0	628354.4	634512.5	627516.0	630127.6	0.996
960	601134.5	613619.3	610111.7	608288.5	610673.2	0.996

Table C.12 Byte throughputs, $s = 1.143$, 100 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	0.040	0.040	0.040	0.040	0.040	1.000
10	0.072	0.072	0.072	0.072	0.072	1.000
20	0.183	0.107	0.135	0.142	0.128	1.108
40	0.238	0.354	0.354	0.315	0.341	0.924
60	0.435	0.420	0.417	0.424	0.420	1.009
100	0.784	0.798	0.730	0.771	0.766	1.006
120	1.014	0.964	0.953	0.977	0.965	1.013
140	1.123	1.227	1.277	1.209	1.238	0.977
200	3.153	3.236	3.207	3.199	3.214	0.995
260	4.914	4.978	4.956	4.949	4.961	0.998
320	6.812	6.801	6.786	6.800	6.796	1.001
400	9.096	9.079	9.163	9.113	9.118	0.999
480	11.318	11.354	11.337	11.336	11.342	0.999
640	15.756	15.733	15.725	15.738	15.732	1.000
800	20.007	20.010	20.044	20.020	20.025	1.000
960	24.109	23.939	23.983	24.010	23.977	1.001

Table C.13 Average latencies, $s = 1.7145$, 100 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	0.288	0.288	0.288	0.288	0.288	1.000
10	2.190	2.190	2.190	2.190	2.190	1.000
20	5.218	5.167	5.107	5.164	5.146	1.004
40	9.680	10.153	10.155	9.996	10.101	0.990
60	14.447	13.895	14.457	14.266	14.206	1.004
100	22.260	22.060	21.530	21.950	21.847	1.005
120	22.870	24.327	26.112	24.436	24.958	0.979
140	29.315	25.810	28.357	27.827	27.331	1.018
200	28.988	28.573	28.820	28.794	28.729	1.002
260	29.972	29.918	29.227	29.706	29.617	1.003
320	30.048	29.987	30.187	30.074	30.082	1.000
400	31.527	31.210	30.770	31.169	31.050	1.004
480	31.528	31.037	31.157	31.241	31.145	1.003
640	30.338	30.185	30.315	30.279	30.260	1.001
800	29.218	29.068	29.247	29.178	29.164	1.000
960	28.015	28.208	28.817	28.347	28.457	0.996

Table C.14 Object throughputs, $s = 1.7145$, 100 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	4019.1	4019.1	4019.1	4019.1	4019.1	1.000
10	53786.0	53786.0	53786.0	53786.0	53786.0	1.000
20	133277.3	132444.6	131731.4	132484.4	132220.2	1.002
40	287685.7	297269.3	297326.8	294093.9	296230.0	0.993
60	371425.8	358667.1	353146.3	361079.7	357631.0	1.010
100	586801.4	582797.3	569697.7	579765.5	577420.1	1.004
120	602927.6	635794.5	680572.8	639765.0	652044.1	0.981
140	727752.1	647885.2	707028.5	694222.0	683045.2	1.016
200	736929.0	726591.4	732207.7	731909.4	730236.2	1.002
260	757918.8	758449.7	743851.8	753406.8	751902.8	1.002
320	751643.2	748774.5	752973.3	751130.3	750959.4	1.000
400	793993.9	786159.3	774043.4	784732.2	781645.0	1.004
480	791300.1	780461.5	784003.8	785255.2	783240.2	1.003
640	755988.3	753485.0	755532.5	755002.0	754673.2	1.000
800	730998.1	728639.1	731089.5	730242.2	729990.2	1.000
960	703420.1	706596.0	720161.3	710059.1	712272.1	0.997

Table C.15 Byte throughputs, $s = 1.7145$, 100 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	0.101	0.101	0.101	0.101	0.101	1.000
10	0.145	0.145	0.145	0.145	0.145	1.000
20	0.190	0.190	0.190	0.190	0.190	1.000
40	0.370	0.303	0.305	0.326	0.311	1.047
60	0.473	0.545	0.516	0.511	0.524	0.976
100	1.200	1.408	1.277	1.295	1.327	0.976
120	1.725	1.736	1.723	1.728	1.729	0.999
140	2.634	2.520	2.395	2.516	2.477	1.016
200	4.544	4.810	4.571	4.642	4.674	0.993
260	6.512	6.623	6.495	6.543	6.554	0.998
320	8.703	8.623	8.698	8.675	8.665	1.001
400	11.134	11.133	11.132	11.133	11.133	1.000
480	13.710	13.648	13.465	13.608	13.574	1.003
640	18.198	18.258	18.205	18.220	18.228	1.000
800	22.697	22.630	22.613	22.647	22.630	1.001
960	26.584	26.665	26.627	26.625	26.639	0.999

Table C.16 Average latencies, $s = 2.5$, 100 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	0.288	0.288	0.288	0.288	0.288	1.000
10	2.648	2.648	2.648	2.648	2.648	1.000
20	5.168	5.168	5.168	5.168	5.168	1.000
40	9.155	9.718	9.710	9.528	9.652	0.987
60	14.427	13.982	14.090	14.166	14.079	1.006
100	20.473	19.117	19.385	19.658	19.387	1.014
120	21.638	20.615	21.075	21.109	20.933	1.008
140	22.885	22.668	23.523	23.026	23.072	0.998
200	24.032	22.843	24.450	23.775	23.689	1.004
260	25.318	25.465	25.410	25.398	25.424	0.999
320	25.800	25.548	26.263	25.871	25.894	0.999
400	26.702	26.440	26.528	26.557	26.508	1.002
480	26.487	26.922	27.010	26.806	26.913	0.996
640	26.525	26.675	26.942	26.714	26.777	0.998
800	26.090	26.173	26.103	26.122	26.133	1.000
960	25.495	26.102	25.613	25.737	25.817	0.997

Table C.17 Object throughputs, $s = 2.5$, 100 Mbps

# of UE's	Run I	Run II	Run III	Mean	Mean (10 Mbps)	Ratio (100/10)
1	12040.3	12040.3	12040.3	12040.3	12040.3	1.000
10	121207.1	121207.1	121207.1	121207.1	121207.1	1.000
20	211001.5	211001.5	211001.5	211001.5	211001.5	1.000
40	353855.8	365357.1	365211.0	361474.6	364014.2	0.993
60	495980.2	500754.3	486834.6	494523.0	494037.3	1.001
100	732518.8	667594.1	704449.4	701520.8	691188.1	1.015
120	762766.6	735515.4	745222.5	747834.8	742857.6	1.007
140	786039.5	767061.9	806059.8	786387.1	786502.9	1.000
200	764958.1	734553.8	784268.5	761260.1	760027.5	1.002
260	812257.5	815907.4	814425.8	814196.9	814843.3	0.999
320	858918.1	837361.3	887855.4	861378.3	862198.3	0.999
400	898436.2	891509.4	894020.4	894655.3	893395.0	1.001
480	893213.9	903537.0	905697.0	900816.0	903350.0	0.997
640	890711.2	894397.8	900972.3	895360.5	896910.2	0.998
800	860794.1	879645.1	879624.2	873354.4	877541.2	0.995
960	846861.9	861119.4	849242.5	852407.9	854256.6	0.998

Table C.18 Byte throughputs, $s = 2.5$, 100 Mbps

Appendix D:

Simulation Results

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.037	0.026	0.021	0.037	0.029
10	0.058	0.055	0.059	0.056	0.057
20	0.057	0.060	0.222	0.201	0.135
40	0.080	0.110	0.125	0.137	0.113
60	0.361	0.549	0.133	0.334	0.341
100	0.436	0.329	0.553	0.299	0.404
120	0.501	0.626	0.436	0.729	0.572
140	0.348	0.732	0.376	0.558	0.503
200	0.627	0.642	0.541	0.858	0.668
260	0.677	0.703	0.585	0.771	0.685
320	0.771	0.850	0.863	0.861	0.836
400	0.974	0.880	1.044	1.005	0.976
480	1.100	1.341	1.120	1.087	1.163
640	2.280	1.933	2.049	2.309	2.141
800	3.156	3.357	3.113	3.552	3.294
960	4.420	5.433	5.403	2.863	4.526

Table D.1 Average latencies for RR-G (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.053	0.105	0.073	0.057	0.288
10	0.692	0.690	0.663	0.683	2.728
20	1.312	1.350	1.318	1.327	5.307
40	2.232	2.170	2.167	2.230	8.798
60	3.472	3.092	3.338	3.312	13.213
100	5.763	6.022	5.753	5.585	23.123
120	6.387	6.462	6.763	6.540	26.152
140	7.705	7.643	7.735	7.542	30.625
200	10.307	10.225	10.245	10.380	41.157
260	14.462	14.928	14.163	14.628	58.182
320	17.397	17.490	17.583	17.182	69.652
400	21.297	21.170	21.083	21.613	85.163
480	24.668	25.290	24.558	24.282	98.798
640	27.858	28.205	28.217	27.847	112.127
800	29.095	28.380	29.113	29.207	115.795
960	29.722	29.437	29.463	29.725	118.347

Table D.2 Object throughputs for RR-G (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	908.8	607.4	308.9	1042.0	2867.1
10	13181.1	10742.3	12606.3	11516.5	48046.2
20	25243.8	26912.2	28944.8	28823.9	109924.6
40	45164.8	41359.4	48847.8	46129.8	181501.8
60	70700.0	68246.0	65476.7	64785.2	269207.9
100	123346.7	132897.7	110590.9	110211.2	477046.5
120	130872.9	134070.0	142093.6	134578.7	541615.2
140	152099.4	156205.0	150152.2	163743.0	622199.7
200	215299.6	214108.2	197029.1	212119.6	838556.5
260	299331.0	308452.1	303665.8	288118.2	1199567.0
320	355366.8	379187.0	373593.6	340085.2	1448232.7
400	449786.8	435167.0	438448.3	441811.1	1765213.1
480	522361.4	506417.6	515784.7	505742.4	2050306.1
640	599848.7	597464.9	573800.9	559681.4	2330795.9
800	590327.0	602284.3	619164.2	596816.1	2408591.6
960	628766.3	606785.4	606475.3	618626.6	2460653.5

Table D.3 Byte throughputs for RR-G (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.032	0.024	0.000	0.000	0.029
10	0.052	0.091	0.057	0.044	0.060
20	0.072	0.071	0.105	0.092	0.084
40	0.118	0.162	0.135	0.103	0.129
60	0.143	0.285	0.262	0.283	0.236
100	0.344	0.447	0.346	0.377	0.376
120	0.475	0.303	0.334	0.557	0.416
140	0.569	0.538	0.520	0.282	0.477
200	0.570	0.607	0.602	0.495	0.569
260	0.990	0.620	0.770	0.527	0.732
320	1.068	0.794	0.780	0.746	0.851
400	1.378	0.860	0.891	1.002	1.040
480	1.857	1.269	1.262	1.368	1.448
640	3.997	2.019	2.015	1.975	2.544
800	4.943	2.747	3.920	3.162	3.728
960	7.967	2.884	5.830	3.428	5.117

Table D.4 Average latencies for XSIZE-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.175	0.113	0.000	0.000	0.288
10	0.782	0.628	0.685	0.625	2.720
20	1.548	1.262	1.293	1.333	5.437
40	2.820	2.480	2.580	2.658	10.538
60	4.675	3.513	3.588	3.732	15.508
100	6.870	5.573	5.618	5.540	23.602
120	7.515	7.108	7.855	7.123	29.602
140	8.867	8.282	6.997	8.205	32.350
200	12.658	11.800	11.045	11.462	46.965
260	14.738	13.802	13.378	13.598	55.517
320	18.150	16.432	16.547	17.085	68.213
400	22.052	20.700	20.085	20.467	83.303
480	25.902	23.780	24.098	23.543	97.323
640	28.010	24.287	25.145	25.827	103.268
800	30.367	27.588	27.168	26.703	111.827
960	30.257	26.597	26.758	27.190	110.802

Table D.5 Object throughputs for XSIZE-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	908.8	607.4	308.9	1042.0	2867.1
10	13181.1	10742.3	12606.3	11516.5	48046.2
20	25243.8	26912.2	28944.8	28823.9	109924.6
40	45164.8	41359.4	48847.8	46129.8	181501.8
60	70700.0	68246.0	65476.7	64785.2	269207.9
100	123346.7	132897.7	110590.9	110211.2	477046.5
120	130872.9	134070.0	142093.6	134578.7	541615.2
140	152099.4	156205.0	150152.2	163743.0	622199.7
200	215299.6	214108.2	197029.1	212119.6	838556.5
260	299331.0	308452.1	303665.8	288118.2	1199567.0
320	355366.8	379187.0	373593.6	340085.2	1448232.7
400	449786.8	435167.0	438448.3	441811.1	1765213.1
480	522361.4	506417.6	515784.7	505742.4	2050306.1
640	599848.7	597464.9	573800.9	559681.4	2330795.9
800	590327.0	602284.3	619164.2	596816.1	2408591.6
960	628766.3	606785.4	606475.3	618626.6	2460653.5

Table D.6 Byte throughputs for XSIZE-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.022	0.046	0.026	0.024	0.029
10	0.040	0.044	0.090	0.057	0.059
20	0.074	0.071	0.087	0.060	0.076
40	0.291	0.094	0.238	0.216	0.219
60	0.127	0.556	0.537	0.211	0.369
100	0.617	0.352	0.606	0.383	0.466
120	0.399	0.296	0.493	0.490	0.417
140	0.400	0.365	0.498	0.426	0.423
200	0.671	0.604	0.668	0.719	0.663
260	0.650	0.628	0.648	0.611	0.634
320	0.820	0.712	0.720	0.817	0.764
400	0.932	0.989	0.972	0.963	0.963
480	1.315	1.281	1.206	1.081	1.217
640	2.038	2.213	2.039	1.992	2.071
800	3.244	3.339	3.329	3.135	3.261
960	4.650	4.429	4.671	4.568	4.579

Table D.7 Average latencies for CONN-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.175	0.113	0.000	0.000	0.288
10	0.782	0.628	0.685	0.625	2.720
20	1.548	1.262	1.293	1.333	5.437
40	2.820	2.480	2.580	2.658	10.538
60	4.675	3.513	3.588	3.732	15.508
100	6.870	5.573	5.618	5.540	23.602
120	7.515	7.108	7.855	7.123	29.602
140	8.867	8.282	6.997	8.205	32.350
200	12.658	11.800	11.045	11.462	46.965
260	14.738	13.802	13.378	13.598	55.517
320	18.150	16.432	16.547	17.085	68.213
400	22.052	20.700	20.085	20.467	83.303
480	25.902	23.780	24.098	23.543	97.323
640	28.010	24.287	25.145	25.827	103.268
800	30.367	27.588	27.168	26.703	111.827
960	30.257	26.597	26.758	27.190	110.802

Table D.8 Object throughputs for CONN-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	192.6	1644.4	452.2	577.9	2867.1
10	6812.4	1745.9	5799.8	14479.4	28837.6
20	38820.9	15131.9	37942.2	13059.5	104954.5
40	49447.5	39882.8	38196.2	66810.7	194337.2
60	58300.9	65110.3	81087.2	84292.4	288790.8
100	95938.6	146134.6	87709.8	113912.6	443695.6
120	139876.5	152364.6	168360.2	117256.2	577857.6
140	172086.0	158394.5	170656.6	167206.1	668343.3
200	231085.4	243112.2	249817.1	205189.5	929204.1
260	307903.1	289440.7	326639.1	284748.3	1208731.2
320	311330.7	391914.1	371227.4	354747.3	1429219.5
400	464060.0	405320.7	444400.6	464518.1	1778299.3
480	462940.4	536914.5	474561.9	534939.3	2009356.1
640	600018.5	576001.9	566050.6	560068.0	2302138.9
800	596722.7	614748.3	588265.9	606179.5	2405916.3
960	582129.5	618108.7	610488.0	607605.2	2418331.4

Table D.9 Byte throughputs for CONN-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.022	0.046	0.026	0.024	0.029
10	0.040	0.044	0.090	0.057	0.059
20	0.074	0.071	0.087	0.060	0.076
40	0.291	0.094	0.238	0.216	0.219
60	0.127	0.556	0.537	0.211	0.369
100	0.617	0.352	0.606	0.383	0.466
120	0.399	0.296	0.493	0.490	0.417
140	0.400	0.365	0.498	0.426	0.423
200	0.671	0.604	0.668	0.719	0.663
260	0.650	0.628	0.648	0.611	0.634
320	0.820	0.712	0.720	0.817	0.764
400	0.932	0.989	0.972	0.963	0.963
480	1.315	1.281	1.206	1.081	1.217
640	2.038	2.213	2.039	1.992	2.071
800	3.244	3.339	3.329	3.135	3.261
960	4.650	4.429	4.671	4.568	4.579

Table D.10 Average latencies for RT-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.288	0.000	0.000	0.000	0.288
10	0.805	0.682	0.673	0.567	2.727
20	1.272	1.417	0.758	1.200	4.647
40	3.152	1.852	2.635	2.053	9.692
60	4.137	3.468	2.652	4.573	14.830
100	6.568	5.753	5.263	6.917	24.502
120	6.167	5.635	6.207	7.413	25.422
140	8.495	7.695	6.303	8.540	31.033
200	10.322	10.772	11.850	10.998	43.942
260	13.997	13.990	12.427	15.800	56.213
320	17.803	18.715	17.982	17.143	71.643
400	19.483	20.083	21.015	20.705	81.287
480	23.683	22.413	25.653	23.768	95.518
640	25.903	26.887	27.943	25.593	106.327
800	25.720	25.785	28.182	25.258	104.945
960	27.565	26.213	29.735	26.973	110.487

Table D.11 Object throughputs for RT-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	2867.1	0.0	0.0	0.0	2867.1
10	12734.4	13028.2	12774.4	9509.2	48046.2
20	29486.6	30132.1	13503.1	24487.6	97609.4
40	65188.1	37784.7	53052.7	41917.8	197943.3
60	91341.7	71911.2	47818.2	93474.1	304545.2
100	133597.1	119595.0	107021.7	144556.2	504770.0
120	120836.0	115509.9	133515.5	157998.5	527859.8
140	163605.4	152302.6	132014.6	178746.1	626668.8
200	207107.6	217846.9	238840.0	229206.1	893000.7
260	298324.0	280126.7	254490.5	321055.7	1153997.0
320	382576.5	388639.8	361162.6	354622.9	1487001.8
400	420757.7	414847.1	433538.4	419643.7	1688787.0
480	484619.6	469820.1	524732.8	500805.3	1979977.8
640	538398.0	560521.3	574102.3	539692.4	2212713.9
800	525655.6	530549.9	587494.1	536333.8	2180033.4
960	584104.5	530864.6	617568.6	562626.7	2295164.5

Table D.12 Byte throughputs for RT-G (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.021	0.033	0.027	0.026	0.029
10	0.065	0.045	0.044	0.043	0.049
20	0.049	0.112	0.052	0.058	0.071
40	0.125	0.115	0.089	0.100	0.107
60	0.419	0.356	0.351	0.136	0.313
100	0.386	0.393	0.508	0.604	0.472
120	0.397	0.466	0.522	0.437	0.454
140	0.676	0.333	0.456	0.474	0.485
200	0.531	0.576	0.383	0.599	0.524
260	0.695	0.619	0.755	0.629	0.674
320	0.841	0.764	0.778	0.720	0.775
400	1.114	1.044	1.071	1.030	1.065
480	1.039	1.251	1.150	1.308	1.188
640	1.993	2.751	2.273	1.382	2.103
800	3.242	3.172	3.433	3.315	3.291
960	5.576	3.614	4.683	4.302	4.547

Table D.13 Average latencies for RAN-G (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.038	0.133	0.035	0.082	0.288
10	0.410	0.562	0.453	0.508	1.933
20	1.385	1.672	1.218	1.212	5.487
40	2.190	2.593	2.550	2.428	9.762
60	3.482	3.490	3.888	3.755	14.615
100	5.473	5.557	4.873	5.538	21.442
120	7.425	6.627	6.705	6.548	27.305
140	8.047	7.948	8.202	8.328	32.525
200	10.878	11.502	10.775	11.363	44.518
260	13.575	13.783	13.545	13.258	54.162
320	17.390	16.778	17.358	17.968	69.495
400	20.950	20.505	20.465	20.355	82.275
480	25.260	26.010	24.725	25.835	101.830
640	27.353	28.192	27.762	27.798	111.105
800	29.087	29.143	29.225	29.117	116.572
960	30.365	29.897	29.158	29.768	119.188

Table D.14 Object throughputs for RAN-G (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	186.6	1737.7	269.6	673.1	2867.1
10	6065.1	10204.7	8626.6	9246.4	34142.9
20	26045.2	38493.2	26106.4	23812.8	114457.6
40	47650.4	55607.8	51311.7	44473.3	199043.2
60	67551.3	70438.9	82801.7	79662.4	300454.3
100	115676.1	111231.1	97069.1	114395.3	438371.6
120	155133.8	133744.9	136692.8	137082.5	562653.9
140	163977.3	161689.4	159991.8	170175.9	655834.4
200	210772.9	237031.0	215682.7	240336.2	903822.8
260	282462.5	281065.4	277040.4	270884.0	1111452.4
320	362370.0	339558.4	362495.9	378700.9	1443125.1
400	431416.0	427050.9	418420.7	423492.9	1700380.5
480	520345.5	528699.0	520354.6	546278.1	2115677.2
640	571340.8	586963.4	583542.6	574719.6	2316566.5
800	605511.9	583021.6	626887.7	609458.5	2424879.6
960	634547.0	626591.3	605418.2	609777.1	2476333.6

Table D.15 Byte throughputs for RAN-G (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.035	0.025	0.021	0.035	0.028
10	0.037	0.111	0.036	0.045	0.057
20	0.065	0.092	0.073	0.063	0.073
40	0.093	0.235	0.253	0.232	0.202
60	0.236	0.176	0.141	0.211	0.191
100	0.621	0.300	0.266	0.260	0.360
120	0.402	0.576	0.417	0.329	0.432
140	0.388	0.339	0.571	0.406	0.427
200	0.552	0.625	0.780	0.627	0.647
260	0.558	0.512	0.569	0.601	0.560
320	0.692	0.672	0.710	0.677	0.687
400	0.867	0.838	0.762	0.851	0.830
480	1.163	1.018	0.943	0.972	1.024
640	1.651	2.763	1.385	2.080	1.967
800	3.160	3.383	1.762	4.358	3.161
960	4.999	4.735	3.801	4.113	4.415

Table D.16 Average latencies for RR-S (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.053	0.105	0.073	0.057	0.288
10	0.620	0.602	0.573	0.567	2.362
20	1.240	1.297	1.282	1.273	5.092
40	2.238	2.163	2.165	2.233	8.800
60	3.698	3.652	3.708	3.708	14.767
100	5.895	5.992	6.080	6.112	24.078
120	6.893	7.207	7.260	6.977	28.337
140	8.173	8.303	8.418	7.900	32.795
200	10.843	11.418	11.350	11.200	44.812
260	14.538	14.772	15.313	14.765	59.388
320	17.737	17.957	18.212	17.683	71.588
400	21.627	21.063	21.222	21.678	85.590
480	25.730	25.455	24.965	25.473	101.623
640	28.002	27.787	28.313	28.478	112.580
800	28.515	28.608	29.062	28.687	114.872
960	29.788	29.125	29.197	29.388	117.498

Table D.17 Object throughputs for RR-S (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	908.8	607.4	308.9	1042.0	2867.1
10	9040.6	12092.5	8471.3	11519.1	41123.5
20	22233.4	26832.9	26200.5	30144.6	105411.4
40	45327.0	46230.3	46348.7	43474.6	181380.7
60	75806.5	75066.1	71376.9	79254.3	301503.7
100	127327.7	119331.1	128432.9	124460.6	499552.3
120	143944.5	141724.5	148091.3	150068.4	583828.8
140	167251.7	160787.3	168461.6	166504.2	663004.8
200	216420.5	240423.7	227037.8	227060.6	910942.6
260	290954.6	292166.0	329194.2	311717.1	1224031.9
320	356468.2	370962.1	398962.7	355018.7	1481411.7
400	457047.7	422245.8	435175.0	459085.5	1773554.0
480	517804.3	529278.2	529286.2	531208.3	2107577.0
640	579295.6	583640.2	587330.4	592936.8	2343203.0
800	596801.8	610133.0	599995.9	586773.2	2393703.9
960	613777.6	605899.4	612240.8	610977.1	2442894.9

Table D.18 Byte throughputs for RR-S (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.031	0.024	0.000	0.000	0.028
10	0.059	0.064	0.040	0.040	0.050
20	0.079	0.068	0.080	0.048	0.070
40	0.568	0.315	0.117	0.145	0.293
60	0.360	0.140	0.236	0.153	0.225
100	0.365	0.623	0.272	0.236	0.390
120	0.414	0.406	0.540	0.361	0.437
140	0.733	0.382	0.497	0.430	0.517
200	0.748	0.641	0.609	0.525	0.632
260	0.848	0.473	0.540	0.747	0.653
320	1.065	0.753	0.675	0.726	0.813
400	1.439	0.829	0.915	0.885	1.031
480	1.906	1.071	1.028	1.025	1.270
640	3.401	1.568	2.943	1.416	2.371
800	7.551	1.522	2.650	2.694	3.744
960	10.211	3.745	3.093	2.395	5.087

Table D.19 Average latencies for XSIZE-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.175	0.113	0.000	0.000	0.288
10	0.672	0.437	0.645	0.613	2.367
20	1.330	1.228	1.300	1.060	4.918
40	2.725	2.542	2.445	2.545	10.257
60	3.732	3.483	3.512	3.320	14.047
100	6.668	7.038	5.580	5.340	24.627
120	6.880	7.508	8.153	5.952	28.493
140	8.592	7.523	8.557	7.828	32.500
200	11.677	11.935	10.750	11.312	45.673
260	14.987	14.802	14.198	14.347	58.333
320	19.255	17.173	16.810	16.963	70.202
400	22.587	19.595	20.305	20.005	82.492
480	25.302	23.247	23.195	23.733	95.477
640	28.840	25.858	26.647	25.405	106.750
800	29.800	25.677	26.298	26.325	108.100
960	29.635	25.207	25.593	24.897	105.332

Table D.20 Object throughputs for XSIZE-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	2341.7	525.4	0.0	0.0	2867.1
10	13085.1	7641.4	10750.6	9719.8	41197.0
20	32476.1	24477.3	25440.6	20327.6	102721.7
40	49681.6	58835.9	46612.5	52892.9	208022.9
60	70853.6	65941.6	73175.3	75762.1	285732.6
100	137021.9	149706.2	113623.6	108770.0	509121.7
120	147425.1	154637.5	166119.7	119479.6	587661.9
140	172719.2	149182.4	184455.0	152553.4	658909.9
200	231147.2	246081.8	221799.9	229114.5	928143.3
260	293062.4	308955.0	293349.2	305767.6	1201134.2
320	376731.8	365302.5	361008.8	356445.9	1459488.9
400	460293.3	411590.3	419860.7	412766.2	1704510.5
480	520964.7	481988.6	492135.0	483603.3	1978691.6
640	589595.4	553344.8	543687.3	534097.3	2220724.9
800	610053.1	535569.6	550594.1	552773.9	2248990.6
960	627698.9	516660.7	527551.3	516100.2	2188011.1

Table D.21 Byte throughputs for XSIZE-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.030	0.028	0.020	0.031	0.028
10	0.082	0.066	0.043	0.053	0.062
20	0.098	0.107	0.058	0.084	0.087
40	0.305	0.261	0.613	0.375	0.393
60	0.434	0.280	0.318	0.316	0.334
100	0.372	0.793	0.479	0.576	0.529
120	0.295	0.427	0.337	0.385	0.364
140	0.483	0.588	0.394	0.539	0.501
200	0.653	0.407	0.549	0.393	0.494
260	0.544	0.600	0.611	0.629	0.596
320	0.828	0.807	0.953	0.850	0.856
400	0.832	0.876	0.868	0.921	0.875
480	1.018	0.962	1.084	0.998	1.016
640	1.968	2.019	1.732	1.998	1.928
800	3.265	3.352	3.315	3.269	3.300
960	4.448	4.455	4.508	4.467	4.470

Table D.22 Average latencies for XSIZE-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.067	0.088	0.047	0.087	0.288
10	0.510	0.785	0.290	0.773	2.358
20	1.293	1.440	1.287	1.237	5.257
40	1.858	2.420	2.287	1.280	7.845
60	3.675	4.492	4.027	3.423	15.617
100	6.235	3.767	6.180	6.092	22.273
120	6.152	6.823	7.048	8.508	28.532
140	7.733	8.775	8.528	7.993	33.030
200	11.065	13.265	11.112	12.102	47.543
260	15.183	14.565	15.065	14.823	59.637
320	17.145	19.353	15.983	16.070	68.552
400	22.288	21.653	21.660	22.722	88.323
480	23.113	25.325	26.680	25.698	100.817
640	29.542	28.867	29.973	29.638	118.020
800	27.840	27.602	28.877	29.273	113.592
960	30.160	29.025	29.583	29.587	118.355

Table D.23 Object throughputs for CONN-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	667.6	958.3	190.2	1050.9	2867.1
10	9592.1	13473.3	6500.7	11441.4	41007.5
20	28807.3	28403.4	23905.5	26754.6	107870.8
40	37363.6	55224.6	46748.6	22258.5	161595.4
60	70683.4	98355.4	82159.2	70897.5	322095.5
100	128194.1	75502.4	128930.0	127549.7	460176.2
120	126215.4	142924.3	146104.8	173910.1	589154.6
140	150736.6	185813.7	174652.2	156337.8	667540.4
200	234700.6	261409.4	220481.1	249653.7	966244.8
260	315303.6	300586.6	311579.8	302597.1	1230067.0
320	365409.3	402596.7	322100.0	328391.5	1418497.6
400	457243.9	443195.5	450214.9	473044.0	1823698.4
480	477807.9	529525.6	548994.8	536118.5	2092446.9
640	608326.9	614222.2	621629.0	613025.2	2457203.3
800	589131.9	588380.2	588736.2	598470.3	2364718.6
960	615489.8	605858.7	615015.8	621753.8	2458118.2

Table D.24 Byte throughputs for CONN-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.028	0	0	0	0.028
10	0.039	0.056	0.04	0.042	0.045
20	0.069	0.065	0.07	0.083	0.072
40	0.212	0.2	0.269	0.523	0.293
60	0.232	0.178	0.249	0.174	0.211
100	0.504	0.472	0.342	0.363	0.413
120	0.396	0.329	0.443	0.291	0.365
140	0.411	0.534	0.503	0.692	0.529
200	0.554	0.578	0.465	0.6	0.548
260	0.559	0.609	0.563	0.573	0.577
320	0.646	0.798	0.721	0.824	0.745
400	0.803	0.832	0.891	0.89	0.854
480	0.958	1.061	1.534	0.983	1.14
640	3.082	1.895	1.699	1.657	2.097
800	3.243	2.762	4.849	2.855	3.463
960	6.105	4.049	4.014	3.755	4.483

Table D.25 Average latencies for RT-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.288	0.000	0.000	0.000	0.288
10	0.630	0.752	0.328	0.658	2.368
20	1.307	1.180	1.417	1.227	5.130
40	2.628	2.802	2.020	2.278	9.728
60	3.180	3.505	4.258	3.192	14.135
100	5.485	5.265	6.983	6.045	23.778
120	6.253	6.938	7.070	6.638	26.900
140	9.052	8.185	8.438	7.722	33.397
200	11.053	12.313	12.562	11.382	47.310
260	14.543	17.025	15.290	13.813	60.672
320	18.487	15.758	17.245	17.553	69.043
400	21.832	20.943	21.605	20.453	84.833
480	23.168	23.882	25.302	24.625	96.977
640	29.545	26.880	27.958	28.593	112.977
800	26.960	27.315	29.620	26.288	110.183
960	29.240	28.975	29.140	29.133	116.488

Table D.26 Object throughputs for RT-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	2867.1	0.0	0.0	0.0	2867.1
10	9346.9	14450.7	5449.6	11952.1	41199.4
20	30526.3	22600.4	27191.3	25906.6	106224.5
40	56278.6	54134.6	39800.8	48341.4	198555.4
60	62788.6	76001.0	86959.0	64201.9	289950.5
100	113721.2	108574.3	145334.6	126250.3	493880.2
120	136178.3	143449.2	137608.1	136183.9	553419.5
140	186196.2	168236.2	161669.9	161782.1	677884.4
200	225082.8	239621.8	257251.3	238182.6	960138.4
260	298069.8	360394.4	308293.8	285559.9	1252317.9
320	385617.1	323488.1	363780.3	364378.0	1437263.6
400	452674.1	428287.0	441946.6	435166.5	1758074.2
480	496824.0	495837.8	515956.0	502465.6	2011083.4
640	602300.4	548219.1	594215.0	605122.0	2349856.5
800	548641.0	572293.2	609305.1	557286.9	2287526.2
960	586039.6	596513.1	622269.2	617404.8	2422226.5

Table D.27 Byte throughputs for RT-S (4-server), $s = 1.143$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.021	0.045	0.027	0.030	0.028
10	0.038	0.063	0.044	0.043	0.047
20	0.068	0.073	0.103	0.073	0.079
40	0.094	0.237	0.443	0.359	0.287
60	0.199	0.262	0.293	0.130	0.221
100	0.250	0.340	0.427	0.397	0.352
120	0.796	0.641	0.426	0.621	0.622
140	0.543	0.423	0.604	0.533	0.526
200	0.392	0.456	0.371	0.522	0.434
260	0.528	0.598	0.498	0.627	0.562
320	0.762	0.710	0.680	0.697	0.713
400	0.867	0.924	0.929	0.801	0.881
480	1.024	1.024	0.944	1.070	1.016
640	1.688	1.437	2.119	2.455	1.928
800	3.248	3.990	3.318	2.418	3.246
960	5.727	3.326	3.372	5.177	4.408

Table D.28 Average latencies for RAN-S (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.108	0.037	0.038	0.105	0.288
10	0.387	0.403	0.493	0.310	1.593
20	1.338	1.438	1.390	1.247	5.413
40	2.075	2.082	2.173	2.285	8.615
60	3.365	3.743	3.478	3.518	14.105
100	5.235	5.435	4.848	5.053	20.572
120	7.048	6.945	6.860	6.995	27.848
140	8.065	8.227	8.300	8.475	33.067
200	11.742	11.010	11.413	11.243	45.408
260	15.413	15.052	15.052	14.470	59.987
320	18.607	18.070	18.020	18.268	72.965
400	21.332	21.605	22.123	21.345	86.405
480	25.973	24.977	24.808	25.265	101.023
640	28.240	27.865	28.428	28.503	113.037
800	28.885	29.518	28.963	29.160	116.527
960	29.982	29.598	29.493	29.870	118.943

Table D.29 Object throughputs for RAN-S (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	377.7	932.8	327.6	1229.0	2867.1
10	5711.5	6185.4	8345.2	6797.6	27039.7
20	30494.0	29213.1	28313.0	24955.3	112975.5
40	37606.4	45807.7	46452.2	47383.3	177249.5
60	64414.7	74910.2	70297.8	79343.8	288966.5
100	108024.9	113415.4	97231.9	103568.9	422241.1
120	144981.2	138983.8	144038.2	148684.5	576687.8
140	160728.2	166146.3	172657.7	169783.4	669315.7
200	242740.7	211868.8	239954.1	228554.8	923118.4
260	324763.2	298729.6	313881.9	300381.2	1237755.9
320	367803.5	386385.4	382991.6	379603.6	1516784.2
400	440334.8	432499.0	481149.5	434540.1	1788523.4
480	528355.9	504806.4	520294.1	540002.7	2093459.2
640	592756.2	555130.9	608604.0	592622.6	2349113.8
800	592043.6	619951.3	605738.0	607249.8	2424982.8
960	621815.1	627354.6	593685.1	627001.0	2469855.8

Table D.30 Byte throughputs for RAN-S (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.147	0.132	0.070	0.061	0.108
10	0.113	0.182	0.132	0.114	0.134
20	0.147	0.174	0.229	0.142	0.173
40	0.181	0.381	0.380	0.181	0.278
60	0.297	0.343	0.288	0.411	0.335
100	0.464	0.612	0.630	0.448	0.539
120	0.625	0.586	0.643	0.596	0.611
140	0.692	0.720	0.603	0.521	0.634
200	0.696	0.722	0.750	0.765	0.733
260	0.930	0.789	0.910	0.895	0.881
320	1.111	1.229	1.227	1.155	1.181
400	1.507	1.550	1.668	1.709	1.609
480	2.104	2.392	2.007	2.206	2.175
640	2.227	5.046	2.707	5.093	3.782
800	5.455	5.316	5.825	5.383	5.494
960	7.150	6.396	5.913	7.879	6.838

Table D.31 Average latencies for RR-G (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.097	0.063	0.075	0.053	0.288
10	0.580	0.552	0.562	0.623	2.317
20	1.165	1.147	1.260	1.355	4.927
40	2.050	1.993	2.045	2.232	8.320
60	3.652	3.655	3.623	3.767	14.697
100	5.640	5.682	5.792	5.725	22.838
120	7.003	7.655	7.252	7.818	29.728
140	8.188	8.060	7.967	8.003	32.218
200	11.340	11.562	11.302	11.227	45.430
260	14.365	14.688	14.578	14.468	58.100
320	17.015	17.160	17.348	17.290	68.813
400	19.922	19.452	19.668	20.312	79.353
480	21.507	21.163	22.045	21.517	86.232
640	21.695	22.210	21.875	22.257	88.037
800	22.828	22.295	22.363	22.838	90.325
960	23.412	23.340	23.522	23.765	94.038

Table D.32 Object throughputs for RR-G (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	7514.7	3934.2	321.8	269.6	12040.2
10	25396.7	46233.3	23520.4	17096.4	112246.8
20	30466.4	92628.8	41236.2	37210.3	201541.6
40	81402.5	75597.8	52609.4	92884.4	302494.1
60	92332.0	194583.1	106818.6	123865.9	517599.6
100	223662.0	172605.0	214924.7	186563.1	797754.8
120	244536.0	279482.3	257357.6	328206.5	1109582.5
140	369253.7	276462.2	271787.5	239243.4	1156746.8
200	356960.2	638939.2	361649.1	394043.4	1751591.9
260	633891.9	542995.1	454033.4	499853.9	2130774.3
320	528368.2	734918.3	729466.9	823636.7	2816390.1
400	610056.5	689560.2	886052.6	915293.7	3100963.0
480	777953.5	879156.8	689021.5	990232.5	3336364.3
640	640513.9	905646.2	697763.8	1139223.5	3383147.3
800	813639.7	1163592.6	645610.7	815676.4	3438519.4
960	1070920.3	814359.5	885171.9	962258.1	3732709.8

Table D.33 Byte throughputs for RR-G (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.110	0.104	0.000	0.000	0.108
10	0.138	0.106	0.131	0.135	0.127
20	0.159	0.599	0.598	0.162	0.365
40	0.207	0.449	0.413	0.265	0.334
60	0.526	0.403	0.862	0.240	0.486
100	0.673	0.618	0.509	0.598	0.600
120	0.513	0.504	0.621	0.536	0.544
140	0.870	0.622	0.726	0.539	0.695
200	0.918	0.911	0.898	0.727	0.866
260	1.635	1.046	0.877	0.932	1.149
320	1.705	1.194	1.132	1.023	1.278
400	2.950	1.521	1.398	1.961	2.003
480	3.964	2.677	1.740	2.487	2.766
640	6.016	3.800	3.714	3.447	4.317
800	8.327	7.184	4.299	3.996	6.057
960	11.497	4.583	3.405	9.762	7.562

Table D.34 Average latencies for XSIZE-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.193	0.095	0.000	0.000	0.288
10	0.745	0.692	0.623	0.603	2.663
20	1.450	1.197	1.207	1.285	5.138
40	2.683	2.530	2.675	2.368	10.257
60	3.748	3.790	3.035	4.002	14.575
100	6.242	5.465	6.108	4.450	22.265
120	6.798	6.568	6.967	7.325	27.658
140	9.267	8.803	7.250	7.835	33.155
200	12.407	10.682	10.222	10.533	43.843
260	15.992	13.290	13.433	12.618	55.333
320	18.293	16.010	16.108	16.190	66.602
400	20.378	17.120	17.022	18.152	72.672
480	21.325	18.525	18.202	18.698	76.750
640	23.143	20.153	19.807	19.480	82.583
800	23.727	20.980	20.432	20.165	85.303
960	24.773	20.415	20.360	21.808	87.357

Table D.35 Object throughputs for XSIZE-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	8192.4	3847.8	0.0	0.0	12040.3
10	28954.6	21670.3	35711.4	35382.7	121719.0
20	65413.4	35966.2	46144.2	62489.1	210013.0
40	95188.5	94950.8	79322.9	107621.5	377083.7
60	127598.9	148935.6	88866.2	133811.1	499211.8
100	205286.9	200548.9	230987.1	147304.0	784126.9
120	271856.9	340120.1	215052.3	214669.4	1041698.8
140	391514.1	237784.8	238277.5	316332.9	1183909.3
200	466135.9	339053.8	430580.6	305175.4	1540945.6
260	651214.2	454871.5	532480.3	408142.9	2046708.9
320	718663.7	601087.8	649775.7	776179.8	2745706.9
400	899157.2	627725.0	572916.4	826207.3	2926005.9
480	757845.6	657247.9	741712.7	873211.9	3030018.0
640	973215.5	789681.2	717983.5	735865.3	3216745.4
800	750241.8	1093471.0	627204.6	835394.6	3306312.0
960	1066657.8	707480.4	676591.3	909400.7	3360130.2

Table D.36 Byte throughputs for XSIZE-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.070	0.175	0.073	0.074	0.108
10	0.164	0.168	0.122	0.216	0.167
20	0.149	0.161	0.168	0.146	0.154
40	0.249	0.192	0.315	0.178	0.227
60	0.733	0.379	1.097	0.285	0.611
100	0.347	0.718	0.425	0.463	0.479
120	0.828	0.712	0.510	0.685	0.675
140	0.640	0.621	0.574	0.646	0.621
200	0.913	0.742	0.977	0.766	0.846
260	0.921	1.011	0.993	0.939	0.964
320	1.220	1.284	1.219	1.168	1.222
400	1.494	1.483	1.581	1.535	1.524
480	2.113	2.201	2.100	2.316	2.181
640	3.843	4.149	4.019	3.997	4.000
800	5.272	5.565	5.450	5.307	5.398
960	6.847	6.825	6.807	6.791	6.817

Table D.37 Average latencies for CONN-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.038	0.098	0.092	0.060	0.288
10	0.683	0.630	0.678	0.640	2.632
20	1.017	0.422	1.143	1.877	4.458
40	1.852	3.077	2.085	2.390	9.403
60	3.055	4.160	3.095	2.795	13.105
100	6.760	5.250	4.853	5.837	22.700
120	6.470	6.905	7.957	6.493	27.825
140	7.752	8.477	7.743	9.257	33.228
200	10.605	12.072	10.762	10.765	44.203
260	15.530	13.685	14.443	15.428	59.087
320	16.742	16.172	16.862	17.363	67.138
400	20.217	19.053	20.272	20.608	80.150
480	21.333	21.502	22.728	21.103	86.667
640	22.258	21.282	20.868	21.977	86.385
800	22.583	22.700	22.730	23.052	91.065
960	23.470	23.293	23.562	23.593	93.918

Table D.38 Object throughputs for CONN-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	185.1	10436.7	454.7	963.7	12040.3
10	48339.3	21350.7	20864.5	30418.8	120973.3
20	30594.6	17322.7	87660.8	53888.1	189466.2
40	67897.3	106155.1	88063.0	96727.9	358843.3
60	127411.8	137960.3	100105.4	98477.4	463954.9
100	221590.4	176885.8	221737.4	178234.9	798448.5
120	235691.4	224216.2	298944.7	297214.1	1056066.3
140	234334.9	307989.1	258369.8	389633.1	1190326.9
200	474955.4	360374.4	368376.5	347070.0	1550776.3
260	534165.6	444207.2	648966.2	532856.5	2160195.6
320	771662.2	554708.7	701221.8	724246.5	2751839.3
400	842338.7	643998.3	919669.2	724375.8	3130382.0
480	853305.8	957919.3	693214.9	838759.2	3343199.2
640	804444.1	711897.8	937488.2	882080.6	3335910.6
800	826149.0	902575.5	780653.8	1044684.8	3554063.0
960	1001744.4	841033.6	913342.1	975529.7	3731649.7

Table D.39 Byte throughputs for CONN-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.108	0.000	0.000	0.000	0.108
10	0.152	0.115	0.138	0.165	0.142
20	0.165	0.157	0.600	0.605	0.349
40	0.545	0.328	0.225	0.534	0.394
60	0.312	0.393	0.247	0.224	0.290
100	0.485	0.489	0.568	0.756	0.559
120	0.690	0.613	0.677	0.546	0.630
140	0.692	0.634	0.750	0.661	0.683
200	0.712	1.003	0.759	0.801	0.815
260	0.913	1.004	0.958	0.849	0.932
320	1.213	1.283	1.301	1.184	1.245
400	2.132	1.506	1.589	1.548	1.703
480	3.405	2.293	1.933	1.816	2.370
640	6.869	2.449	3.444	3.145	4.036
800	10.698	4.273	3.477	3.112	5.629
960	12.715	5.023	5.121	4.535	7.064

Table D.40 Average latencies for RT-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.288	0.000	0.000	0.000	0.288
10	0.760	0.675	0.653	0.565	2.653
20	1.507	1.517	1.075	1.173	5.272
40	2.238	2.225	2.980	2.247	9.690
60	3.737	3.242	4.302	3.490	14.770
100	5.812	7.402	6.218	4.578	24.010
120	7.153	7.377	5.477	6.690	26.697
140	7.190	7.897	7.767	8.435	31.288
200	12.307	10.955	11.462	11.232	45.955
260	15.112	15.335	14.700	14.187	59.333
320	17.233	17.012	17.140	16.697	68.082
400	20.943	19.520	18.857	19.110	78.430
480	21.092	20.705	20.393	20.473	82.663
640	22.685	20.587	21.025	21.542	85.838
800	24.927	21.243	20.910	20.877	87.957
960	24.868	21.370	21.720	21.632	89.590

Table D.41 Object throughputs for RT-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	12040.3	0.0	0.0	0.0	12040.3
10	47768.9	19515.2	22785.5	31151.0	121220.5
20	58851.2	72578.3	24310.8	58821.2	214561.6
40	90008.9	115436.4	95464.9	63915.1	364825.3
60	118238.4	134127.5	134429.0	117103.4	503898.3
100	171354.7	239906.1	209361.3	205715.3	826337.5
120	344776.7	224987.4	209133.2	230903.7	1009801.0
140	227006.1	363902.2	269928.3	264064.3	1124900.9
200	400494.2	466882.8	506776.5	389471.1	1763624.7
260	426669.8	657634.3	517178.5	572006.0	2173488.7
320	735307.0	690164.8	831065.6	523426.1	2779963.5
400	1034647.7	781620.3	678952.1	583234.5	3078454.5
480	840696.6	677887.3	706708.0	991001.2	3216293.1
640	825194.9	614667.8	900629.2	978029.1	3318521.0
800	749605.8	886955.2	835355.5	894477.8	3366394.4
960	859925.9	824134.7	1035619.9	699292.9	3418973.3

Table D.42 Byte throughputs for RT-G (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.070	0.161	0.136	0.058	0.108
10	0.239	0.138	0.119	0.101	0.151
20	0.161	0.163	0.167	0.150	0.161
40	0.187	0.487	0.478	0.281	0.357
60	0.530	0.517	0.460	0.326	0.462
100	0.342	0.527	0.838	0.513	0.560
120	0.791	0.716	0.774	0.424	0.670
140	0.519	0.577	0.566	0.516	0.545
200	0.831	0.771	0.693	0.798	0.774
260	0.968	0.901	0.862	0.818	0.888
320	1.188	1.008	1.107	1.194	1.126
400	1.534	1.505	1.808	1.708	1.638
480	1.815	2.363	3.080	1.943	2.308
640	5.703	2.544	2.702	4.220	3.803
800	5.755	6.192	4.723	4.981	5.420
960	6.208	5.338	8.322	7.493	6.864

Table D.43 Average latencies for RAN-G (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.112	0.077	0.065	0.035	0.288
10	0.668	0.628	0.525	0.702	2.523
20	1.073	1.295	1.087	1.092	4.547
40	2.230	2.160	2.305	2.418	9.113
60	3.992	3.760	3.880	3.495	15.127
100	5.265	6.027	5.873	5.998	23.163
120	6.338	6.038	6.080	6.850	25.307
140	7.625	7.922	7.865	7.647	31.058
200	11.743	11.445	11.325	11.663	46.177
260	14.588	14.592	14.123	14.688	57.992
320	17.072	16.372	17.328	17.662	68.433
400	19.913	19.922	19.500	19.885	79.220
480	20.617	21.607	21.697	21.130	85.050
640	22.355	21.390	22.313	21.613	87.672
800	22.643	23.100	22.310	22.563	90.617
960	22.608	23.032	23.843	24.195	93.678

Table D.44 Object throughputs for RAN-G (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	1142.9	6859.3	3983.3	54.8	12040.3
10	56397.7	30190.9	14429.2	18145.3	119163.1
20	34796.2	45597.0	65459.0	45177.6	191029.8
40	88357.9	83067.6	89797.1	75560.2	336782.7
60	152874.7	145997.3	141527.6	89810.8	530210.4
100	158190.1	178340.2	266541.5	206971.1	810042.9
120	238731.4	204814.8	175319.9	243505.1	862371.2
140	253056.7	290701.0	214346.4	262167.8	1020271.7
200	400475.4	507034.3	492295.8	370593.3	1770398.7
260	641241.3	558433.5	447284.3	481963.2	2128922.4
320	702583.9	709468.7	611433.2	772311.5	2795797.2
400	587029.1	778161.4	998138.9	740791.2	3104120.5
480	699864.9	802358.6	964101.2	828847.7	3295172.4
640	1032374.9	892717.1	808010.9	640190.8	3373293.7
800	866803.9	1008555.8	748773.8	918548.7	3542682.2
960	739880.4	836492.6	1135071.6	1014424.7	3725869.2

Table D.45 Byte throughputs for RAN-G (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.142	0.127	0.070	0.060	0.105
10	0.786	0.804	0.152	0.155	0.464
20	0.222	0.157	0.270	0.187	0.210
40	0.505	0.236	0.174	0.972	0.482
60	0.362	0.442	0.239	0.308	0.338
100	0.590	0.621	0.497	0.619	0.583
120	0.583	0.556	0.468	0.502	0.527
140	0.461	0.504	0.504	0.446	0.478
200	0.815	0.925	0.790	0.626	0.787
260	0.762	0.889	0.783	0.863	0.824
320	1.060	1.004	1.164	1.193	1.106
400	1.530	1.414	1.494	1.555	1.498
480	2.028	1.821	1.781	2.937	2.143
640	4.114	3.944	2.583	4.527	3.788
800	5.847	4.987	4.933	5.661	5.360
960	7.303	6.511	6.676	6.320	6.705

Table D.46 Average latencies for RR-S (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.097	0.063	0.075	0.053	0.288
10	0.612	0.573	0.607	0.652	2.443
20	1.265	1.277	1.382	1.457	5.380
40	2.035	1.972	2.042	2.178	8.227
60	3.542	3.628	3.578	3.573	14.322
100	6.230	6.283	5.997	6.330	24.840
120	7.112	7.197	7.115	7.070	28.493
140	8.495	8.572	8.322	8.712	34.100
200	11.742	11.383	11.602	11.938	46.665
260	15.053	15.300	15.052	14.855	60.260
320	16.555	16.487	16.700	17.178	66.920
400	20.375	20.503	20.623	20.307	81.808
480	21.888	21.600	22.163	22.047	87.698
640	21.662	21.813	21.728	21.302	86.505
800	22.627	22.423	22.237	22.728	90.015
960	23.707	23.635	23.685	23.247	94.273

Table D.47 Object throughputs for RR-S (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	7514.7	3934.2	321.8	269.6	12040.2
10	31977.8	18256.4	30003.1	33889.7	114127.0
20	42662.1	35184.6	60415.0	80412.2	218674.0
40	94278.4	57007.1	82283.8	66471.5	300040.9
60	132879.3	113587.0	127505.5	119645.8	493617.6
100	181877.2	246744.9	191975.4	229872.3	850469.7
120	245338.3	399102.5	206151.9	223525.5	1074118.2
140	400860.8	254210.4	273302.5	291724.8	1220098.4
200	543052.6	470598.7	322705.4	449568.3	1785925.0
260	541855.2	501740.7	614370.4	540514.4	2198480.8
320	479554.7	835197.4	806047.8	634280.4	2755080.3
400	995957.5	832368.8	710322.0	635430.5	3174078.8
480	875462.4	817752.0	800503.1	882840.1	3376557.6
640	818734.2	984100.3	771993.1	764843.8	3339671.4
800	1011147.2	971796.0	706717.7	836825.0	3526486.0
960	909845.5	900035.6	1132045.0	796541.9	3738468.1

Table D.48 Byte throughputs for RR-S (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.126	0.067	0.000	0.000	0.105
10	0.163	0.104	0.151	0.147	0.142
20	0.154	0.155	0.169	0.138	0.153
40	0.183	0.220	0.310	0.397	0.280
60	0.322	0.237	0.238	0.207	0.252
100	0.775	0.830	0.631	0.481	0.676
120	0.648	0.524	0.613	0.501	0.570
140	0.768	0.555	0.611	0.531	0.628
200	1.042	0.779	0.738	0.877	0.861
260	1.901	0.723	0.802	0.846	1.130
320	2.373	2.426	0.991	0.894	1.775
400	3.453	3.938	3.298	0.916	3.062
480	5.485	2.668	3.365	1.243	3.429
640	7.387	6.983	1.420	4.013	5.290
800	11.970	5.363	1.225	6.215	6.709
960	17.397	5.643	3.619	1.896	8.385

Table D.49 Average latencies for XSIZE-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.185	0.103	0.000	0.000	0.288
10	0.738	0.655	0.687	0.573	2.653
20	1.300	1.382	1.118	1.425	5.225
40	2.090	2.300	2.208	2.327	8.925
60	3.587	3.572	3.735	3.172	14.065
100	5.557	5.995	5.163	6.340	23.055
120	7.837	7.637	6.338	7.670	29.482
140	10.398	8.508	8.775	7.223	34.905
200	11.975	11.908	10.952	10.232	45.067
260	16.197	11.767	12.102	13.065	53.130
320	16.238	17.775	12.890	12.680	59.583
400	17.197	17.183	15.225	11.832	61.437
480	21.712	16.945	16.663	14.018	69.338
640	20.092	19.905	14.093	16.653	70.743
800	21.537	19.948	13.972	19.870	75.327
960	25.553	18.753	17.788	14.502	76.597

Table D.50 Object throughputs for XSIZE-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	11653.2	387.1	0.0	0.0	12040.3
10	47125.6	13704.2	20322.4	40068.3	121220.5
20	61145.3	40729.3	40005.4	71443.0	213323.0
40	80902.5	86526.9	92406.1	56128.6	315964.2
60	115459.4	121548.1	148104.2	101494.9	486606.6
100	204709.1	267256.8	142019.8	191246.3	805232.1
120	305282.3	267726.9	294328.0	233382.4	1100719.6
140	329652.7	342288.9	322105.2	244452.0	1238498.8
200	335597.8	442193.0	469945.4	493029.8	1740766.2
260	787990.0	349088.1	412695.0	414399.5	1964172.7
320	619027.5	967788.0	405268.7	334255.9	2326340.0
400	564213.5	673971.8	602646.6	645825.5	2486657.4
480	974313.3	726747.1	719909.0	409797.0	2830766.3
640	789748.6	880659.1	529120.7	674561.4	2874089.7
800	668877.2	871050.5	459751.6	994703.3	2994382.6
960	1262314.2	669680.9	644593.4	447446.7	3024035.1

Table D.51 Byte throughputs for XSIZE-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.129	0.111	0.100	0.068	0.105
10	0.118	0.176	0.139	0.142	0.144
20	0.184	0.183	0.186	0.132	0.171
40	0.272	0.228	0.190	0.221	0.229
60	0.282	0.406	0.454	0.324	0.367
100	0.467	0.597	0.501	0.544	0.522
120	0.823	0.683	0.630	0.514	0.651
140	0.662	0.627	0.464	0.662	0.602
200	0.701	0.657	0.727	0.869	0.735
260	0.789	0.793	0.825	0.847	0.812
320	1.091	1.096	1.066	1.130	1.095
400	1.541	1.473	1.443	1.538	1.498
480	2.065	1.898	2.111	2.004	2.020
640	3.814	3.984	3.872	3.852	3.880
800	5.557	5.318	5.322	5.377	5.393
960	6.729	6.717	6.787	6.939	6.792

Table D.52 Average latencies for XSIZE-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.067	0.085	0.088	0.048	0.288
10	0.658	0.662	0.653	0.677	2.650
20	1.093	1.500	1.480	1.410	5.483
40	2.852	3.055	2.725	2.172	10.803
60	2.778	4.115	2.828	4.058	13.780
100	6.783	4.965	6.552	6.085	24.385
120	5.718	7.328	7.530	7.813	28.390
140	8.618	7.610	8.318	7.460	32.007
200	12.423	11.612	11.815	10.690	46.540
260	14.848	15.810	15.352	13.130	59.140
320	17.373	17.042	18.030	16.895	69.340
400	19.593	20.853	19.515	20.052	80.013
480	21.128	21.660	22.077	21.733	86.598
640	22.183	21.520	21.812	22.053	87.568
800	22.648	22.590	23.142	21.863	90.243
960	23.415	23.648	23.318	23.017	93.398

Table D.53 Object throughputs for CONN-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	4456.3	3931.9	3485.2	166.9	12040.3
10	20288.8	39008.4	33115.4	28794.7	121207.3
20	46043.0	89312.3	44450.4	40743.8	220549.5
40	127827.9	112060.9	99911.8	53642.5	393443.1
60	75471.5	154851.0	80893.5	166996.4	478212.4
100	196956.7	171259.2	246983.5	220561.5	835760.9
120	243673.8	238410.1	254596.9	337913.3	1074594.1
140	314935.4	259146.9	255379.6	212965.0	1042426.8
200	416427.9	469955.5	362373.0	364871.9	1613628.2
260	480339.7	510207.5	474200.8	693776.2	2158524.2
320	664066.9	895462.9	763560.9	509747.0	2832837.6
400	859520.7	753383.0	563682.4	945762.9	3122348.9
480	632115.9	976008.0	736647.8	1000348.7	3345120.3
640	721645.7	996449.3	873928.5	776520.9	3368544.3
800	727629.4	979940.3	699219.3	1034828.2	3441617.2
960	865634.0	1282803.7	763745.7	805848.1	3718031.5

Table D.54 Byte throughputs for CONN-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.105	0.000	0.000	0.000	0.105
10	0.136	0.167	0.166	0.130	0.149
20	0.222	0.138	0.165	0.170	0.175
40	0.186	0.196	0.198	0.335	0.230
60	0.244	0.276	0.255	0.244	0.254
100	0.394	0.570	0.613	0.398	0.489
120	0.685	0.520	0.705	0.605	0.626
140	0.560	0.666	0.667	0.706	0.648
200	0.698	0.652	0.607	0.683	0.658
260	0.930	0.825	0.879	0.753	0.847
320	1.073	1.057	1.099	1.117	1.087
400	1.527	1.586	1.509	1.449	1.519
480	2.269	2.111	1.911	2.119	2.103
640	3.999	3.844	3.238	4.285	3.837
800	5.064	5.426	5.661	5.261	5.356
960	5.334	5.386	10.734	5.660	6.826

Table D.55 Average latencies for RT-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.288	0.000	0.000	0.000	0.288
10	0.710	0.613	0.525	0.542	2.390
20	1.502	1.375	1.172	1.132	5.180
40	2.227	2.975	2.105	2.515	9.822
60	3.788	3.400	3.637	4.008	14.833
100	5.445	6.803	4.603	6.622	23.473
120	7.563	7.437	6.213	7.060	28.273
140	8.765	9.007	8.297	7.848	33.917
200	11.343	12.027	12.908	12.073	48.352
260	14.570	14.707	16.027	14.955	60.258
320	17.292	17.653	17.665	18.027	70.637
400	18.782	20.537	20.538	18.628	78.485
480	21.518	21.820	21.482	20.988	85.808
640	21.473	22.123	22.712	22.187	88.495
800	21.917	22.973	22.938	23.293	91.122
960	22.735	22.873	23.827	22.480	91.915

Table D.56 Object throughputs for RT-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	12040.3	0.0	0.0	0.0	12040.3
10	33622.9	46103.6	19535.5	14191.9	113453.8
20	74390.3	49036.9	30218.1	57525.7	211171.0
40	73256.6	105639.4	87609.5	100484.9	366990.4
60	162020.7	110623.0	132156.7	117011.9	521812.3
100	211692.3	223290.1	159307.9	219499.5	813789.9
120	263445.5	340149.4	193617.5	270570.4	1067782.7
140	278162.7	306545.4	397682.3	233084.6	1215475.1
200	461961.2	579988.5	407147.6	376896.1	1825993.4
260	448831.6	566279.2	532066.8	647621.3	2194798.9
320	711689.9	587408.7	792571.5	775843.9	2867514.1
400	875598.4	718433.8	903494.2	585152.4	3082678.8
480	947253.6	754588.4	780096.4	837155.3	3319093.6
640	782775.8	827141.9	791549.1	990168.4	3391635.2
800	1056043.8	842478.6	808917.2	761802.2	3469241.8
960	1001462.7	811113.9	867695.6	809718.8	3489991.0

Table D.57 Byte throughputs for RT-S (4-server), $s = 2.5$, averaging window = 1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.115	0.075	0.140	0.117	0.105
10	0.127	0.150	0.097	0.134	0.129
20	0.153	0.139	0.755	0.722	0.448
40	0.222	0.189	0.426	0.678	0.386
60	0.334	0.322	0.342	0.543	0.384
100	0.708	0.736	0.453	0.452	0.593
120	0.803	0.464	0.843	0.738	0.718
140	0.627	0.472	0.697	0.600	0.600
200	0.762	0.710	0.673	0.662	0.702
260	0.784	0.855	0.817	0.837	0.824
320	1.038	1.227	0.973	1.152	1.099
400	1.463	1.371	1.700	1.647	1.546
480	1.591	2.135	2.153	2.911	2.201
640	4.330	4.916	2.221	3.601	3.778
800	4.742	5.528	5.999	5.292	5.398
960	6.625	6.915	6.801	6.610	6.738

Table D.58 Average latencies for RAN-S (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.068	0.105	0.047	0.068	0.288
10	0.688	0.758	0.540	0.677	2.663
20	1.373	1.213	1.352	1.325	5.263
40	2.507	2.307	2.327	2.633	9.773
60	3.755	4.447	3.610	3.902	15.713
100	5.702	5.877	5.267	5.292	22.137
120	6.853	6.283	6.990	7.028	27.155
140	8.840	8.282	8.592	8.668	34.382
200	11.690	11.828	11.733	11.918	47.170
260	15.050	16.192	14.628	15.055	60.925
320	17.457	17.977	17.382	17.660	70.475
400	19.517	20.095	20.005	20.277	79.893
480	20.712	20.540	20.803	21.093	83.148
640	22.123	22.303	21.660	21.848	87.935
800	22.322	23.098	23.298	22.550	91.268
960	23.548	23.267	23.850	23.490	94.155

Table D.59 Object throughputs for RAN-S (4-server), $s = 2.5$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	3253.7	1088.0	3887.4	3811.1	12040.3
10	35005.1	39795.0	15381.5	31537.4	121719.0
20	68892.2	40941.3	60580.6	43902.9	214317.0
40	91006.5	65192.2	111153.5	98904.8	366257.1
60	132866.1	155207.1	111190.1	148641.6	547905.0
100	206771.5	236544.2	161986.3	175737.9	781039.9
120	248345.6	208135.4	227793.2	239289.9	923564.1
140	366640.3	263047.4	331542.0	263755.3	1224985.0
200	306439.0	383942.0	720821.4	383468.4	1794670.8
260	478455.1	532286.5	653246.3	550390.9	2214378.7
320	697211.2	601011.2	819206.7	742343.4	2859772.5
400	776159.2	772412.0	921867.1	649500.0	3119938.3
480	664805.8	889801.2	759234.8	891846.7	3205688.5
640	661892.3	897183.5	856494.4	963288.1	3378858.2
800	708955.5	1046110.7	966277.7	841376.7	3562720.6
960	906506.2	1081139.5	990291.7	756772.8	3734710.2

Table D.60 Byte throughputs for RAN-S (4-server), $s = 2.5$

#of UE's	Server I	Server II	Mean
1	0.028	0.030	0.029
10	0.055	0.046	0.050
20	0.085	0.086	0.086
40	0.302	0.305	0.304
60	0.307	0.417	0.363
100	0.407	0.673	0.543
120	0.356	0.416	0.386
140	0.442	0.446	0.444
200	0.665	0.690	0.678
260	1.041	1.031	1.036
320	1.824	1.690	1.757
400	2.448	3.615	3.032
480	4.460	3.731	4.092
640	6.873	6.041	6.455
800	10.286	7.167	8.719
960	11.655	10.288	10.969

Table D.61 Average latencies for RR-G (2-server), $s = 1.143$

#of UE's	Server I	Server II	Mean
1	0.127	0.162	0.288
10	1.192	1.175	2.367
20	2.728	2.780	5.508
40	4.750	4.785	9.535
60	7.333	7.450	14.783
100	10.310	10.725	21.035
120	13.102	13.235	26.337
140	15.118	15.163	30.282
200	20.810	20.782	41.592
260	26.088	26.653	52.742
320	28.170	28.282	56.452
400	28.492	28.532	57.023
480	28.753	29.283	58.037
640	31.055	31.347	62.402
800	30.763	31.062	61.825
960	31.242	31.510	62.752

Table D.62 Object throughputs for RR-G (2-server), $s = 1.143$

#of UE's	Server I	Server II	Mean
1	1217.7	1649.4	2867.1
10	20203.6	20993.4	41197.0
20	56990.6	57800.1	114790.7
40	94908.6	99770.8	194679.5
60	146160.6	155662.7	301823.3
100	209760.0	221412.9	431172.9
120	265019.6	278806.1	543825.7
140	301014.0	313348.9	614362.9
200	422837.3	420203.3	843040.6
260	533671.0	542892.6	1076563.6
320	579015.8	575181.7	1154197.5
400	586639.7	578473.2	1165112.9
480	597247.6	589372.3	1186619.9
640	641916.1	637482.6	1279398.7
800	639112.5	629249.1	1268361.6
960	645923.1	640356.5	1286279.6

Table D.63 Byte throughputs for RR-G (2-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Server V	Server VI	Server VII	Server VIII	Mean
1	0.044	0.025	0.023	0.024	0.030	0.027	0.018	0.059	0.029
10	0.070	0.070	0.042	0.073	0.036	0.038	0.044	0.048	0.054
20	0.649	0.058	0.607	0.082	0.044	0.068	0.049	0.078	0.215
40	0.102	0.094	0.104	0.084	0.071	0.101	0.121	0.084	0.095
60	0.092	0.092	0.105	0.540	0.096	0.536	0.291	0.383	0.265
100	0.748	0.438	0.480	0.462	0.445	0.732	0.253	0.279	0.480
120	0.474	0.572	0.579	0.421	0.435	0.718	0.726	0.417	0.544
140	0.178	0.657	0.210	0.355	0.755	0.510	0.302	0.575	0.441
200	0.557	0.562	0.455	0.448	0.700	0.560	0.582	0.589	0.557
260	0.439	0.824	0.655	0.730	0.676	0.679	0.649	0.531	0.648
320	0.606	0.753	0.674	0.684	0.759	0.721	0.807	0.785	0.724
400	0.866	0.833	0.768	0.840	0.826	0.902	0.894	0.990	0.864
480	0.955	1.055	0.912	1.055	0.833	0.903	0.896	0.891	0.937
640	1.198	1.175	1.103	1.156	1.198	1.043	1.152	1.086	1.139
800	1.390	1.541	1.306	1.412	1.389	1.420	1.365	1.406	1.403
960	1.755	1.564	1.495	1.564	1.668	1.551	1.608	1.538	1.593

Table D.64 Average latencies for RR-G (8-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Server V	Server VI	Server VII	Server VIII	Mean
1	0.025	0.078	0.047	0.037	0.028	0.027	0.027	0.020	0.288
10	0.303	0.327	0.292	0.293	0.262	0.242	0.233	0.223	2.175
20	0.613	0.617	0.658	0.585	0.582	0.575	0.533	0.583	4.747
40	1.012	0.940	0.983	0.955	0.877	0.903	0.837	0.903	7.410
60	1.565	1.675	1.605	1.695	1.740	1.555	1.563	1.518	12.917
100	2.855	2.900	2.767	2.895	2.850	2.732	2.752	2.783	22.533
120	3.440	3.438	3.565	3.380	3.302	3.528	3.375	3.432	27.460
140	3.815	3.807	3.953	3.793	3.895	3.938	3.937	3.502	30.640
200	5.868	5.833	5.692	5.873	5.795	5.825	5.837	5.598	46.322
260	7.247	7.268	7.062	7.313	7.260	6.980	7.470	7.333	57.933
320	8.552	9.143	8.862	9.060	8.857	9.282	8.828	9.098	71.682
400	10.315	10.245	10.670	10.617	10.892	10.487	10.495	10.383	84.103
480	13.040	12.997	13.153	12.825	12.937	12.950	13.185	12.877	103.963
640	17.403	17.263	17.395	17.417	17.248	17.297	17.702	16.927	138.652
800	20.688	20.548	20.980	20.705	21.005	20.867	20.810	20.757	166.360
960	24.878	24.173	24.347	24.302	24.232	24.298	24.290	24.138	194.658

Table D.65 Object throughputs for RR-G (8-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Server V	Server VI	Server VII	Server VIII	Mean
1	570.8	449.4	258.7	245.6	338.0	157.9	50.2	796.5	2867.1
10	4597.8	5446.2	5981.1	4940.6	4031.5	3216.5	4277.7	5280.7	37772.0
20	11542.4	10590.4	19386.3	12459.4	10557.9	12381.0	11619.2	11212.0	99748.6
40	21615.4	19122.5	18462.4	18045.3	15666.4	19484.5	20252.8	21027.4	153676.8
60	33357.3	34329.7	32984.5	33121.7	32691.0	30121.7	34003.1	36185.3	266794.1
100	64039.2	57054.5	55210.0	64902.8	55712.7	55716.4	53984.9	57896.5	464517.1
120	74326.1	70978.5	73974.8	68082.9	66618.1	70396.6	75589.2	69828.6	569794.8
140	76113.8	76717.0	83686.5	73909.0	75219.5	83179.8	81469.9	71493.3	621788.9
200	120670.3	112501.7	113936.9	121684.7	117866.5	114650.5	126205.5	114316.3	941832.4
260	146183.4	145752.8	141499.5	153727.8	149848.4	151086.6	150695.5	151378.2	1190172.2
320	168406.9	204206.3	178301.7	190224.1	184904.9	188703.4	184537.3	188851.0	1488135.6
400	214109.7	208045.2	223528.5	232747.7	212165.8	216387.8	226430.7	211905.4	1745320.8
480	265656.3	276038.8	275735.4	268396.4	256546.2	273544.5	286031.0	257765.9	2159714.4
640	351983.8	356588.0	358507.6	366874.3	368640.0	357021.4	367084.4	352444.5	2879144.0
800	420300.2	427640.5	430645.9	429416.1	450313.8	432475.1	450315.3	429429.1	3470536.2
960	514337.8	515282.3	532450.2	494563.3	504098.8	514087.4	493752.4	498277.4	4066849.5

Table D.66 Byte throughputs for RR-G (8-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.029	0.000	0.000	0.000	0.029
10	0.079	0.040	0.042	0.042	0.053
20	0.072	0.092	0.095	0.145	0.105
40	0.120	0.147	0.139	0.097	0.124
60	0.682	0.358	0.145	0.203	0.350
100	0.501	0.365	0.308	0.425	0.400
120	0.647	0.562	0.339	0.297	0.461
140	0.864	0.621	0.680	0.553	0.679
200	0.687	0.637	0.590	0.589	0.627
260	0.744	0.649	0.737	0.772	0.724
320	0.779	0.823	0.907	0.822	0.832
400	0.943	0.852	0.930	0.981	0.927
480	1.170	1.154	1.274	1.198	1.200
640	2.301	3.199	1.683	1.703	2.225
800	5.511	2.644	3.425	1.758	3.376
960	7.607	3.042	3.244	4.378	4.631

Table D.67 Average latencies for XSIZE-G (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.288	0.000	0.000	0.000	0.288
10	0.843	0.712	0.630	0.542	2.727
20	0.462	1.737	1.705	1.337	5.240
40	3.227	1.780	2.652	2.380	10.038
60	3.450	3.820	3.135	3.647	14.052
100	6.508	5.015	6.648	5.477	23.648
120	7.368	6.582	7.287	7.122	28.358
140	7.770	8.082	7.995	7.765	31.612
200	11.640	11.725	11.182	10.872	45.418
260	13.600	13.823	12.742	12.612	52.777
320	18.320	18.087	17.607	17.507	71.520
400	21.107	20.827	21.323	21.393	84.650
480	26.232	24.112	26.342	23.677	100.362
640	28.013	27.767	27.840	27.045	110.665
800	30.397	27.240	29.015	28.653	115.305
960	30.637	28.335	27.893	29.450	116.315

Table D.68 Object throughputs for XSIZE-G (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	2867.1	0.0	0.0	0.0	2867.1
10	15517.3	10731.6	11609.8	10167.8	48026.5
20	8038.7	35486.1	38060.1	26091.5	107676.5
40	59142.2	39946.5	59150.2	45275.7	203514.6
60	72134.7	69766.8	64398.3	79450.3	285750.1
100	135715.8	105141.0	132367.7	114921.5	488146.0
120	150195.8	137507.4	153680.3	142845.6	584229.0
140	150671.1	168754.5	167815.1	149291.7	636532.5
200	243157.2	233580.0	225325.4	221964.0	924026.6
260	279905.7	282975.9	260395.6	255754.8	1079032.0
320	382937.1	363881.0	377174.4	358839.7	1482832.1
400	442822.8	430483.5	449358.4	434351.1	1757015.7
480	536185.1	492800.9	543981.5	506728.8	2079696.4
640	584934.3	579021.8	575875.9	559535.5	2299367.5
800	636104.0	583268.4	600436.1	581680.6	2401489.1
960	641928.4	596238.3	592666.8	588937.8	2419771.2

Table D.69 Byte throughputs for XSIZE-G (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.035	0.025	0.022	0.000	0.029
10	0.068	0.059	0.074	0.044	0.062
20	0.685	0.046	0.092	0.753	0.391
40	0.117	0.287	0.266	0.239	0.225
60	0.244	0.276	0.495	0.270	0.317
100	0.283	0.419	0.414	0.443	0.382
120	0.311	0.604	0.577	0.375	0.458
140	0.575	0.296	0.376	0.667	0.467
200	0.782	0.512	0.639	0.617	0.636
260	0.923	0.742	0.743	0.650	0.754
320	1.194	1.111	0.937	0.993	1.058
400	1.557	1.352	1.414	1.183	1.394
480	2.293	1.940	1.605	1.215	1.786
640	3.615	2.284	1.827	2.892	2.699
800	5.826	3.881	3.203	2.528	3.929
960	7.470	3.427	5.115	4.353	5.190

Table D.70 Average latencies for XSIZE-G (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.113	0.158	0.017	0.000	0.288
10	0.797	0.675	0.668	0.582	2.722
20	1.295	1.115	1.295	1.083	4.788
40	2.652	2.410	2.492	2.478	10.032
60	4.293	3.567	3.560	3.847	15.267
100	6.932	4.910	6.900	4.637	23.378
120	8.153	7.688	5.505	6.785	28.132
140	8.678	8.352	8.840	6.428	32.298
200	11.593	11.907	10.673	12.688	46.862
260	11.837	15.792	13.402	15.812	56.842
320	16.345	18.223	18.707	14.812	68.087
400	21.865	19.995	22.298	15.125	79.283
480	23.658	21.760	22.385	19.805	87.608
640	28.207	25.753	23.070	25.780	102.810
800	29.615	26.292	25.057	26.210	107.173
960	30.828	25.710	27.088	27.385	111.012

Table D.71 Object throughputs for XSIZE-G (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	1837.2	967.9	62.0	0.0	2867.1
10	13113.1	12949.5	11960.4	9584.4	47607.3
20	27674.0	19528.1	31819.2	21333.5	100354.8
40	59171.3	43844.3	49389.0	50985.6	203390.3
60	77035.2	75315.9	76018.8	84200.1	312570.1
100	142814.4	101663.7	143858.6	91971.1	480307.8
120	165959.0	154517.4	115976.7	143508.0	579961.2
140	166695.6	172729.0	184165.7	130908.7	654498.9
200	227346.4	246269.6	222978.6	255294.6	951889.2
260	248847.1	325981.8	273414.5	319282.2	1167525.7
320	322076.6	378092.6	394271.8	314467.5	1408908.5
400	440885.9	420657.9	462414.1	319381.1	1643339.0
480	483184.6	451397.5	465654.8	411260.0	1811497.0
640	570960.7	538920.8	493362.0	530267.7	2133511.1
800	597498.4	550219.7	516309.6	566609.8	2230637.4
960	653877.5	523302.9	564823.3	569834.1	2311838.0

Table D.72 Byte throughputs for XSIZE-G (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.039	0.024	0.022	0.000	0.029
10	0.044	0.043	0.060	0.062	0.052
20	0.053	0.073	0.073	0.092	0.072
40	0.116	0.234	0.115	0.974	0.392
60	0.491	0.609	0.176	0.380	0.421
100	0.505	0.244	0.430	0.614	0.436
120	0.409	0.337	0.528	0.431	0.427
140	0.514	0.498	0.586	0.765	0.587
200	0.629	0.560	0.823	0.548	0.647
260	1.071	0.654	0.845	0.896	0.871
320	1.254	0.838	0.798	1.022	0.977
400	1.821	1.183	1.240	1.360	1.401
480	2.447	1.392	1.453	1.773	1.790
640	3.582	2.479	2.571	2.413	2.782
800	5.377	2.739	2.612	5.460	4.176
960	8.102	3.925	5.084	3.911	5.372

Table D.73 Average latencies for XSIZE-G (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.098	0.145	0.045	0.000	0.288
10	0.777	0.610	0.640	0.705	2.732
20	1.282	1.233	1.325	1.192	5.032
40	2.513	2.513	2.010	2.835	9.872
60	3.883	3.820	3.387	3.835	14.925
100	5.660	6.402	5.422	4.957	22.440
120	7.778	7.032	7.373	6.527	28.710
140	8.507	7.768	8.287	7.382	31.943
200	14.003	11.960	11.715	7.737	45.415
260	12.997	12.073	14.673	15.487	55.230
320	16.780	19.188	15.057	15.353	66.378
400	19.768	20.698	18.570	20.025	79.062
480	23.597	18.823	22.512	22.418	87.350
640	27.027	24.817	24.502	24.135	100.480
800	28.397	22.663	23.028	26.857	100.945
960	29.553	24.862	26.955	25.278	106.648

Table D.74 Object throughputs for XSIZE-G (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	1927.0	713.6	226.4	0.0	2867.1
10	12503.4	10358.3	12094.7	13588.9	48545.3
20	22280.4	26933.2	28426.0	27109.9	104749.5
40	44755.8	50350.8	42932.0	62776.6	200815.3
60	81861.7	82579.2	63755.1	78370.0	306566.1
100	117906.3	127006.0	121377.3	94640.5	460930.1
120	167428.0	137649.8	145991.8	142349.3	593418.9
140	161679.6	158641.9	162917.8	162070.5	645309.7
200	270613.4	252068.7	245745.9	153572.9	922000.9
260	257343.6	246123.4	300916.1	321230.4	1125613.5
320	335322.5	398559.5	312118.6	327965.1	1373965.7
400	404236.3	427704.3	384322.3	419942.0	1636204.8
480	489551.0	389551.5	458894.4	467911.5	1805908.5
640	550379.7	510461.7	522191.0	503344.4	2086376.7
800	591455.0	479732.0	480530.9	545773.1	2097491.0
960	600897.6	526762.7	575024.8	518453.4	2221138.5

Table D.75 Byte throughputs for XSIZE-G (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.026	0.029	0.024	0.033	0.029
10	0.037	0.071	0.043	0.041	0.050
20	0.163	0.047	0.096	0.130	0.123
40	0.160	0.240	0.105	0.193	0.177
60	0.349	0.448	0.211	0.148	0.264
100	0.594	0.246	0.459	0.381	0.427
120	0.357	0.569	0.545	0.359	0.449
140	0.479	0.404	0.692	0.688	0.553
200	0.570	0.524	0.553	0.593	0.560
260	0.601	0.646	0.682	0.570	0.623
320	0.916	0.775	0.756	0.838	0.816
400	0.957	0.967	0.941	0.847	0.927
480	1.220	1.142	1.179	1.150	1.172
640	2.063	1.915	2.209	2.076	2.064
800	3.324	3.272	3.335	3.392	3.330
960	4.571	4.326	4.652	4.540	4.522

Table D.76 Average latencies for CONN-G (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.035	0.122	0.057	0.075	0.288
10	0.260	0.732	0.595	0.778	2.365
20	1.585	0.393	1.660	1.650	5.288
40	1.523	2.712	2.503	2.255	8.993
60	4.300	2.018	4.497	4.323	15.138
100	5.653	4.507	5.742	6.842	22.743
120	7.168	5.447	6.438	6.695	25.748
140	8.408	9.358	7.170	7.652	32.588
200	9.865	10.970	10.730	11.743	43.308
260	15.217	11.208	15.502	15.970	57.897
320	14.842	18.432	19.570	18.527	71.370
400	23.162	21.700	20.850	22.953	88.665
480	22.693	25.475	26.237	24.027	98.432
640	27.915	29.728	27.898	28.825	114.367
800	28.798	29.607	29.862	28.578	116.845
960	29.025	29.560	29.260	30.003	117.848

Table D.77 Object throughputs for CONN-G (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	231.6	1243.7	273.0	1118.7	2867.1
10	4940.4	12899.8	10884.4	12456.4	41181.0
20	36767.0	7450.2	35275.1	29439.2	108931.4
40	31113.7	56723.0	55378.2	41358.5	184573.4
60	83571.4	39994.4	102071.6	87311.6	312949.0
100	115614.0	84531.7	116196.9	151802.2	468144.7
120	145574.9	115324.3	129199.0	140560.1	530658.3
140	167050.1	191455.4	147360.8	154868.8	660735.1
200	195619.2	225847.6	219057.9	242158.0	882682.8
260	299933.9	231467.5	327895.7	330867.4	1190164.5
320	310129.0	384277.0	405285.0	382779.6	1482470.8
400	476408.4	450122.9	435042.6	469085.3	1830659.3
480	469628.8	539996.6	528454.1	501685.2	2039764.8
640	583536.2	631748.8	563356.0	600576.8	2379217.7
800	599338.2	614994.0	613481.6	602623.7	2430437.6
960	603451.8	636072.3	603072.1	607033.2	2449629.4

Table D.78 Byte throughputs for CONN-G (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.029	0.029	0.063	0.022	0.029
10	0.030	0.045	0.040	0.044	0.043
20	0.921	0.094	0.855	0.066	0.376
40	0.501	0.685	0.124	0.146	0.321
60	0.219	0.225	0.181	0.268	0.224
100	0.216	0.520	0.409	0.521	0.405
120	0.429	0.735	0.559	0.629	0.581
140	0.614	0.465	0.467	0.494	0.505
200	0.521	0.635	0.644	0.449	0.555
260	0.747	0.759	0.710	0.732	0.736
320	0.650	0.791	0.786	0.718	0.738
400	1.068	0.970	1.203	0.997	1.057
480	1.243	1.331	1.273	1.315	1.290
640	2.120	2.031	2.204	1.980	2.085
800	3.475	3.239	3.212	3.439	3.341
960	4.664	4.728	4.433	4.420	4.561

Table D.79 Average latencies for CONN-G (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.033	0.108	0.023	0.123	0.288
10	0.110	0.875	0.473	0.797	2.255
20	0.818	1.683	0.922	1.325	4.748
40	2.197	1.772	2.270	3.395	9.633
60	4.222	2.657	3.943	4.158	14.980
100	6.765	4.720	6.312	6.000	23.797
120	7.580	6.775	7.878	5.620	27.853
140	6.737	8.442	7.587	8.693	31.458
200	12.225	10.442	9.228	11.832	43.727
260	14.465	12.928	14.745	13.415	55.553
320	16.922	18.768	17.262	18.232	71.183
400	22.107	23.103	20.282	20.247	85.738
480	25.493	25.367	24.792	24.737	100.388
640	28.345	26.973	28.538	27.483	111.340
800	28.782	29.672	29.042	29.258	116.753
960	29.372	28.892	30.037	28.295	116.595

Table D.80 Object throughputs for CONN-G (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	221.4	1235.2	995.4	415.1	2867.1
10	687.0	16690.0	7594.4	14609.3	39580.7
20	15603.3	33588.8	24103.2	26434.1	99729.4
40	44355.5	35455.4	47126.9	69765.8	196703.6
60	82046.9	54467.3	81653.9	89186.4	307354.5
100	137008.7	94993.4	132182.0	129291.0	493475.1
120	153591.8	147201.9	161999.7	109818.7	572612.1
140	142684.5	160176.6	157932.5	176173.0	636966.6
200	250226.0	209752.3	191698.5	238084.1	889760.9
260	297458.1	271427.2	299778.5	273038.4	1141702.3
320	357918.1	377837.1	368259.0	373929.1	1477943.3
400	455963.5	475395.2	424551.1	418092.8	1774002.5
480	516258.9	522095.2	526500.0	518014.1	2082868.2
640	579895.0	567812.6	596932.6	569644.5	2314284.6
800	599726.9	626986.8	599672.9	602614.7	2429001.3
960	627319.8	595689.6	616206.9	585827.0	2425043.2

Table D.81 Byte throughputs for CONN-G (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.027	0.024	0.037	0.027	0.029
10	0.058	0.059	0.044	0.039	0.050
20	0.413	0.429	0.450	0.713	0.517
40	0.141	0.238	0.706	0.239	0.274
60	0.292	0.136	0.158	0.284	0.217
100	0.567	0.263	0.411	0.287	0.375
120	0.367	0.274	0.477	0.477	0.387
140	0.655	0.571	0.767	0.561	0.638
200	0.635	0.559	0.537	0.589	0.581
260	0.668	0.546	0.710	0.694	0.657
320	0.907	0.779	0.844	0.913	0.862
400	1.212	1.150	1.145	1.264	1.192
480	1.454	1.485	1.501	1.834	1.571
640	2.497	2.561	2.422	2.266	2.438
800	4.071	3.429	3.333	3.227	3.513
960	4.987	4.205	5.096	4.114	4.613

Table D.82 Average latencies for CONN-G (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.062	0.088	0.080	0.058	0.288
10	0.612	0.748	0.550	0.665	2.575
20	1.107	0.355	1.685	1.327	4.473
40	3.080	2.965	1.437	3.165	10.647
60	3.038	3.498	3.878	4.188	14.603
100	5.228	6.163	6.840	6.537	24.768
120	7.668	8.832	8.447	4.498	29.445
140	8.005	8.402	8.248	8.463	33.118
200	12.263	10.848	11.495	10.975	45.582
260	13.495	14.183	15.813	15.280	58.772
320	17.555	15.215	18.820	15.087	66.677
400	21.363	21.475	18.688	18.422	79.948
480	21.885	21.442	23.270	22.950	89.547
640	26.222	26.388	26.623	25.740	104.973
800	27.542	29.093	28.050	27.563	112.248
960	30.075	27.827	29.687	28.540	116.128

Table D.83 Object throughputs for CONN-G (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	572.6	380.8	1408.1	505.5	2867.1
10	11475.2	14626.4	9351.6	9774.7	45227.9
20	20252.7	6011.2	39268.0	28122.9	93654.8
40	63636.5	58211.1	27132.4	67416.6	216396.6
60	62682.8	69076.7	81036.2	86458.3	299253.9
100	105522.4	131707.0	140267.4	135257.1	512753.8
120	154821.3	182844.5	170971.7	97386.9	606024.4
140	167856.5	169334.7	163908.6	168845.5	669945.3
200	243415.6	229308.7	234028.2	221216.2	927968.7
260	279050.0	285263.2	333541.8	315575.1	1213430.1
320	378495.7	307268.4	390164.5	304465.8	1380394.4
400	440128.2	436225.9	385265.6	401573.3	1663193.1
480	447017.2	429611.7	481228.5	493750.2	1851607.6
640	561602.2	531653.2	563482.2	526976.8	2183714.5
800	571374.6	598912.5	585601.3	582155.8	2338044.3
960	629717.9	579962.2	602148.3	605772.4	2417600.7

Table D.84 Byte throughputs for CONN-G (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.028	0.000	0.000	0.000	0.028
10	0.065	0.053	0.073	0.060	0.062
20	0.134	0.082	0.097	0.127	0.111
40	0.321	0.310	0.129	0.154	0.243
60	0.114	0.169	0.278	0.167	0.171
100	0.395	0.294	0.322	0.317	0.329
120	0.779	0.810	0.498	0.429	0.606
140	0.379	0.629	0.367	0.382	0.425
200	0.551	0.516	0.517	0.573	0.538
260	0.601	0.724	0.519	0.624	0.617
320	0.686	0.708	0.658	0.717	0.692
400	0.850	0.758	0.873	0.888	0.844
480	0.963	0.903	0.983	0.918	0.943
640	2.010	2.044	1.900	1.702	1.915
800	3.244	3.080	3.345	3.393	3.268
960	4.502	4.315	4.638	4.336	4.447

Table D.85 Average latencies for RT-S (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.288	0.000	0.000	0.000	0.288
10	0.788	0.757	0.618	0.557	2.720
20	1.512	1.342	1.115	1.355	5.323
40	2.638	2.483	1.343	2.435	8.900
60	3.677	4.600	2.298	4.160	14.735
100	5.500	6.768	6.020	6.425	24.713
120	6.607	5.538	8.772	7.550	28.467
140	8.252	5.962	7.488	9.115	30.817
200	11.077	10.343	11.705	9.515	42.640
260	15.393	14.973	14.800	14.253	59.420
320	18.915	18.417	16.855	16.318	70.505
400	22.627	19.913	21.477	22.347	86.363
480	26.342	23.942	26.753	26.635	103.672
640	28.462	28.020	29.272	27.597	113.350
800	28.602	27.470	28.253	29.582	113.907
960	29.905	29.138	29.122	30.237	118.402

Table D.86 Object throughputs for RT-S (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	2867.1	0.0	0.0	0.0	2867.1
10	12531.9	13034.2	11988.3	10052.9	47607.3
20	30299.2	29185.4	24213.4	26719.7	110417.7
40	53112.1	47782.7	26153.7	56214.1	183262.5
60	75802.1	90146.5	46594.5	90517.2	303060.3
100	116912.9	142725.1	121887.5	132192.7	513718.2
120	137348.4	120288.5	177392.1	152863.9	587892.9
140	170357.3	119488.9	150979.3	181989.9	622815.5
200	219890.0	208899.5	244678.7	194247.5	867715.6
260	319622.7	307184.7	297383.6	298457.8	1222648.8
320	397411.3	377030.4	353400.9	338160.0	1466002.7
400	477947.7	395943.5	434757.4	479980.9	1788629.5
480	541386.9	493192.2	545243.4	573021.6	2152844.1
640	597521.7	572949.9	604988.8	583337.9	2358798.3
800	581105.6	578726.3	601270.2	611310.1	2372412.4
960	631384.9	588152.8	596432.7	644812.6	2460783.1

Table D.87 Byte throughputs for RT-S (4-server), $s = 1.143$, averaging window = 0.1s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.028	0.029	0.000	0.000	0.028
10	0.040	0.045	0.041	0.039	0.041
20	0.216	0.074	0.260	0.056	0.154
40	0.839	0.487	0.140	0.457	0.442
60	0.205	0.205	0.241	0.408	0.256
100	0.303	0.673	0.706	0.613	0.557
120	0.455	0.314	0.437	0.379	0.396
140	0.368	0.437	0.333	0.453	0.398
200	0.424	0.427	0.489	0.486	0.456
260	0.562	0.659	0.599	0.687	0.627
320	0.780	0.942	0.958	0.750	0.858
400	1.655	0.876	0.878	1.020	1.121
480	1.896	1.217	1.243	1.376	1.448
640	1.877	2.799	2.976	1.898	2.415
800	2.438	4.130	4.042	3.483	3.535
960	4.373	3.279	3.879	6.822	4.618

Table D.88 Average latencies for RT-S (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.163	0.125	0.000	0.000	0.288
10	0.737	0.458	0.643	0.532	2.370
20	1.435	1.508	1.268	1.067	5.278
40	2.030	1.935	3.047	2.110	9.122
60	4.368	2.853	3.560	2.732	13.513
100	7.357	5.753	5.822	5.790	24.722
120	6.708	6.718	6.590	6.515	26.532
140	7.005	8.483	8.707	8.322	32.517
200	11.658	11.800	11.667	11.393	46.518
260	14.197	13.598	14.578	14.920	57.293
320	15.970	17.303	16.122	16.577	65.972
400	22.283	20.107	20.482	20.035	82.907
480	24.843	21.620	22.088	22.390	90.942
640	23.665	26.687	28.392	25.923	104.667
800	25.870	26.123	28.023	27.257	107.273
960	28.572	28.233	29.275	30.150	116.230

Table D.89 Object throughputs for RT-S (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	1621.4	1245.6	0.0	0.0	2867.1
10	12552.4	9930.8	10948.7	7824.9	41256.8
20	24606.9	35655.4	27503.8	20969.4	108735.6
40	40150.5	44317.0	61396.9	41239.8	187104.3
60	87365.2	59008.4	71079.4	57551.9	275004.9
100	149667.9	125144.3	118956.3	114829.4	508597.9
120	137884.4	132074.7	131734.5	144796.1	546489.6
140	151026.4	167606.6	165481.8	174974.9	659089.7
200	238548.3	245308.3	236549.7	225438.6	945844.8
260	288526.4	287389.2	291642.3	312171.2	1179729.1
320	334783.9	362171.9	327300.8	338739.2	1362995.8
400	452637.5	432505.2	421875.0	415703.2	1722720.9
480	508421.3	444684.0	458038.0	470583.6	1881726.9
640	495516.3	556002.5	579228.8	542620.0	2173367.6
800	531000.9	536721.4	593882.7	568799.4	2230404.5
960	590825.7	584958.9	628680.5	614914.3	2419379.3

Table D.90 Byte throughputs for RT-S (4-server), $s = 1.143$, averaging window = 4s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.034	0.023	0.021	0.000	0.028
10	0.039	0.076	0.042	0.060	0.054
20	0.073	0.114	0.088	0.063	0.083
40	0.240	0.195	0.205	0.313	0.239
60	0.142	0.132	0.491	0.368	0.265
100	0.267	0.472	0.421	0.419	0.394
120	0.652	0.551	0.412	0.464	0.518
140	0.468	0.337	0.439	0.456	0.424
200	0.573	0.559	0.729	0.513	0.595
260	0.566	0.583	0.559	0.571	0.570
320	0.791	0.718	0.851	0.800	0.790
400	1.344	0.995	1.411	1.006	1.199
480	1.889	2.028	1.046	1.988	1.756
640	2.305	2.532	2.650	3.514	2.768
800	2.480	3.224	2.053	7.703	4.103
960	2.579	4.168	4.370	8.628	5.062

Table D.91 Average latencies for RT-S (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.133	0.143	0.012	0.000	0.288
10	0.523	0.642	0.682	0.518	2.365
20	1.320	0.982	1.493	1.322	5.117
40	1.540	2.692	2.527	2.727	9.485
60	4.183	4.425	3.397	3.260	15.265
100	5.867	5.623	5.368	6.383	23.242
120	7.148	6.048	7.495	6.798	27.490
140	8.015	8.102	7.643	7.817	31.577
200	11.935	11.865	11.480	10.517	45.797
260	12.895	13.035	15.907	16.100	57.937
320	17.503	17.577	18.305	16.722	70.107
400	19.553	16.830	21.102	20.303	77.788
480	23.362	20.800	19.647	22.270	86.078
640	23.318	24.058	23.615	25.797	96.788
800	24.397	24.297	23.430	30.265	102.388
960	25.667	25.693	26.155	29.373	106.888

Table D.92 Object throughputs for RT-S (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	2168.5	656.8	41.8	0.0	2867.1
10	8923.4	11903.7	11522.1	8831.7	41181.0
20	22314.3	21278.7	31945.3	30327.7	105866.0
40	34472.6	53127.1	52051.7	54303.8	193955.1
60	90439.6	86606.9	70897.1	64910.0	312853.6
100	114566.0	119006.3	115738.9	127501.4	476812.5
120	141422.6	128438.7	155349.3	140104.2	565314.9
140	155623.3	162965.3	156278.7	161424.0	636291.3
200	239589.4	241310.2	237991.0	211439.8	930330.5
260	263967.5	264874.1	326899.7	337757.5	1193498.8
320	361117.9	359093.2	384488.2	350919.0	1455618.3
400	396625.9	360063.3	436474.4	431061.7	1624225.3
480	493426.5	420129.1	411648.9	457580.4	1782785.0
640	483665.4	492246.4	484180.8	548811.6	2008904.2
800	507332.1	500732.1	490594.3	627915.5	2126573.9
960	538485.6	545452.1	542767.6	598735.1	2225440.4

Table D.93 Byte throughputs for RT-S (4-server), $s = 1.143$, averaging window = 8s

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.029	0.000	0.000	0.000	0.029
10	0.067	0.043	0.041	0.039	0.046
20	0.064	0.073	0.061	0.070	0.066
40	0.133	0.122	0.104	0.124	0.123
60	0.169	0.163	0.359	0.146	0.211
100	0.315	0.563	0.630	0.244	0.440
120	0.403	0.554	0.474	0.658	0.506
140	0.533	0.327	0.523	0.621	0.493
200	0.730	0.559	0.585	0.606	0.622
260	0.962	0.807	0.835	0.631	0.805
320	1.155	0.877	0.693	0.735	0.864
400	1.570	0.987	0.963	1.005	1.134
480	1.739	1.106	1.086	1.154	1.287
640	3.273	2.083	1.974	1.713	2.283
800	5.000	3.245	3.095	3.211	3.674
960	7.205	4.326	4.260	4.032	5.004

Table D.94 Average latencies for Bryhni-CONN-G-1.1 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.288	0.000	0.000	0.000	0.288
10	0.388	0.775	0.618	0.587	2.368
20	1.850	1.257	1.382	0.923	5.412
40	3.225	2.110	1.762	2.815	9.912
60	3.732	4.563	3.955	3.107	15.357
100	4.943	6.027	5.355	5.850	22.175
120	8.655	6.105	7.967	5.595	28.322
140	7.732	8.732	8.305	6.892	31.660
200	12.000	10.820	11.945	10.322	45.087
260	14.780	13.878	14.023	15.862	58.543
320	17.768	16.703	18.277	17.408	70.157
400	20.912	20.268	22.000	18.272	81.452
480	27.537	23.882	25.198	23.278	99.895
640	29.310	25.605	26.062	27.838	108.815
800	29.638	26.988	26.472	26.763	109.862
960	29.090	26.590	26.847	26.827	109.353

Table D.95 Object throughputs for Bryhni-CONN-G-1.1 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	2867.1	0.0	0.0	0.0	2867.1
10	7459.2	14147.1	10325.3	9267.8	41199.4
20	36398.1	26833.3	30833.8	18936.7	113001.9
40	63612.7	42327.1	39784.2	55833.6	201557.6
60	81530.6	91961.6	82031.8	58966.9	314490.9
100	105531.7	125892.6	106727.0	118923.2	457074.4
120	178569.5	128973.8	158215.3	117376.8	583135.4
140	145077.4	177696.7	179340.4	137565.9	639680.3
200	228989.8	215779.7	251613.0	221154.1	917536.6
260	303050.7	287594.6	290673.8	323894.3	1205213.3
320	363659.4	351993.2	376805.2	360602.2	1453059.9
400	439196.9	418186.4	459943.2	375977.2	1693303.7
480	574254.2	495670.5	515775.0	488302.0	2074001.7
640	584735.6	544009.2	556792.2	576887.3	2262424.4
800	596195.3	580266.1	545937.4	560591.9	2282990.8
960	599835.3	553580.2	572388.5	548712.1	2274516.1

Table D.96 Byte throughputs for Bryhni-CONN-G-1.1 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.029	0.000	0.000	0.000	0.029
10	0.054	0.031	0.000	0.000	0.054
20	0.110	0.133	0.000	0.000	0.112
40	0.119	0.109	0.000	0.000	0.117
60	0.247	0.101	0.000	0.000	0.209
100	0.291	0.554	0.050	0.000	0.384
120	0.609	0.466	0.058	0.000	0.547
140	0.440	0.410	0.418	0.000	0.426
200	0.554	0.510	0.594	0.000	0.534
260	0.809	0.643	0.624	0.000	0.723
320	0.834	0.763	0.467	0.000	0.785
400	0.883	0.763	0.657	0.659	0.808
480	0.999	1.014	0.953	0.502	0.996
640	1.280	1.137	1.151	1.134	1.195
800	1.818	1.554	1.357	1.326	1.553
960	2.204	1.966	1.800	1.682	1.927

Table D.97 Average latencies for Bryhni-CONN-G-1.0 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.288	0.000	0.000	0.000	0.288
10	2.622	0.110	0.000	0.000	2.732
20	4.808	0.445	0.000	0.000	5.253
40	7.175	1.860	0.000	0.000	9.035
60	9.995	3.582	0.000	0.000	13.577
100	14.605	8.307	0.138	0.000	23.050
120	14.625	10.948	0.042	0.000	25.615
140	18.445	14.585	0.142	0.000	33.172
200	23.067	21.780	0.855	0.000	45.702
260	27.818	26.412	2.590	0.000	56.820
320	34.633	33.193	3.007	0.000	70.833
400	41.753	37.617	9.368	0.303	89.042
480	48.120	43.428	13.833	0.750	106.132
640	52.633	46.647	26.578	8.508	134.367
800	49.495	44.772	39.387	24.047	157.700
960	49.657	45.703	45.768	39.567	180.695

Table D.98 Object throughputs for Bryhni-CONN-G-1.0 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	2867.1	0.0	0.0	0.0	2867.1
10	46756.2	1312.0	0.0	0.0	48068.3
20	100215.8	7625.3	0.0	0.0	107841.2
40	146816.8	38338.9	0.0	0.0	185155.7
60	206990.2	73196.9	0.0	0.0	280187.0
100	295216.8	179556.2	2193.7	0.0	476966.7
120	302659.0	223639.7	903.9	0.0	527202.6
140	366894.8	300191.7	3286.7	0.0	670373.1
200	469328.2	444135.0	15417.2	0.0	928880.4
260	570375.2	546721.6	54793.8	0.0	1171890.6
320	721440.5	683641.1	63869.5	0.0	1468951.1
400	864247.5	772886.2	193225.9	6398.9	1836758.4
480	1013736.1	895671.7	283982.6	15979.0	2209369.5
640	1082101.8	967652.4	570450.3	176690.0	2796894.5
800	1021665.0	957150.1	804522.8	506957.0	3290294.8
960	1043493.8	967577.9	929606.4	838266.2	3778944.2

Table D.99 Byte throughputs for Bryhni-CONN-G-1.0 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.024	0.033	0.023	0.035	0.029
10	0.041	0.038	0.037	0.057	0.043
20	0.060	0.070	0.063	0.079	0.068
40	0.077	0.432	0.241	0.177	0.228
60	0.224	0.125	0.392	0.167	0.228
100	0.591	0.571	0.382	0.385	0.484
120	0.492	0.317	0.450	0.591	0.463
140	0.448	0.277	0.499	0.477	0.426
200	0.648	0.543	0.563	0.527	0.570
260	0.815	0.537	0.685	0.570	0.651
320	0.767	0.713	0.764	0.782	0.756
400	0.869	0.802	0.893	0.886	0.862
480	1.026	1.050	0.963	0.877	0.978
640	1.101	1.129	1.149	1.189	1.142
800	1.733	1.862	1.353	1.800	1.689
960	2.308	4.739	2.242	2.813	3.051

Table D.100 Average latencies for Bryhni-RT-G-1.0 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.080	0.070	0.070	0.068	0.288
10	0.595	0.600	0.628	0.547	2.370
20	1.328	1.297	1.343	1.370	5.338
40	2.295	2.125	2.238	2.285	8.943
60	3.432	3.543	3.472	3.245	13.692
100	5.905	5.850	5.727	5.747	23.228
120	6.490	6.212	6.332	6.195	25.228
140	8.085	7.895	8.008	7.925	31.913
200	11.317	11.250	10.735	11.410	44.712
260	14.528	15.047	14.550	14.720	58.845
320	18.747	17.430	17.535	17.355	71.067
400	21.805	22.422	21.272	21.625	87.123
480	25.428	25.235	25.530	25.987	102.180
640	33.997	34.598	35.212	35.090	138.897
800	37.628	38.983	37.295	38.098	152.005
960	35.470	37.653	35.257	36.403	144.783

Table D.101 Object throughputs for Bryhni-RT-G-1.0 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	491.0	973.7	344.3	1058.1	2867.1
10	11242.2	10471.0	9500.2	10043.4	41256.8
20	30317.1	23349.3	30074.3	27434.6	111175.4
40	43881.3	44663.4	46994.2	48448.5	183987.4
60	69386.0	69633.0	68956.5	70970.8	278946.3
100	123767.5	120210.7	118061.2	120648.9	482688.2
120	135844.1	132075.2	131424.1	124731.7	524075.0
140	165861.9	159709.4	161503.8	158608.0	645683.1
200	231093.7	233253.7	219546.1	226488.3	910381.8
260	300634.9	308714.9	299962.7	305563.3	1214875.8
320	388695.2	358065.0	364131.4	368325.8	1479217.4
400	449037.0	453811.5	445821.5	451365.6	1800035.6
480	535949.4	520576.2	523811.8	541250.6	2121588.0
640	703894.3	721379.4	733262.3	738245.2	2896781.1
800	789689.3	804978.7	791365.9	783328.3	3169362.2
960	743557.9	782456.9	743715.6	745732.5	3015462.9

Table D.102 Byte throughputs for Bryhni-RT-G-1.0 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.022	0.038	0.000	0.000	0.028
10	0.062	0.046	0.042	0.000	0.054
20	0.706	0.045	0.610	0.046	0.417
40	0.447	0.225	0.230	0.101	0.308
60	0.206	0.291	0.577	0.367	0.334
100	0.376	0.485	0.213	0.710	0.413
120	0.663	0.271	0.429	0.299	0.436
140	0.531	0.369	0.560	0.500	0.485
200	0.587	0.633	0.399	0.590	0.554
260	0.446	0.662	0.621	0.591	0.576
320	0.659	0.707	0.586	0.597	0.640
400	0.632	0.655	0.825	0.780	0.722
480	0.729	0.897	0.748	0.807	0.795
640	1.035	1.176	0.967	0.988	1.044
800	1.799	1.978	1.847	2.122	1.935
960	3.692	2.271	2.504	2.814	2.832

Table D.103 Average latencies for Bryhni-XSIZE-S-1.0 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	0.192	0.097	0.000	0.000	0.288
10	1.347	1.153	0.232	0.000	2.732
20	1.892	1.798	0.817	0.097	4.603
40	3.957	3.413	2.452	0.360	10.182
60	5.397	4.893	3.568	1.218	15.077
100	7.745	7.092	5.813	3.175	23.825
120	9.638	9.352	7.912	4.220	31.122
140	10.437	9.843	7.865	5.987	34.132
200	14.358	13.185	11.573	7.947	47.063
260	17.768	16.385	15.433	13.172	62.758
320	20.635	19.608	17.970	17.633	75.847
400	24.080	21.460	22.658	21.567	89.765
480	27.517	26.742	26.125	25.143	105.527
640	36.807	35.198	33.268	30.758	136.032
800	36.117	35.730	36.963	34.990	143.800
960	38.693	37.178	36.065	34.198	146.135

Table D.104 Object throughputs for Bryhni-XSIZE-S-1.0 (4-server), $s = 1.143$

#of UE's	Server I	Server II	Server III	Server IV	Mean
1	866.4	2000.6	0.0	0.0	2867.1
10	24284.3	18959.6	4824.3	0.0	48068.3
20	39427.3	32145.6	22587.3	2609.2	96769.3
40	74686.0	68604.1	56264.5	6903.1	206457.7
60	110025.7	94751.4	84807.4	22234.2	311818.6
100	161888.3	142681.0	128414.3	61237.2	494220.9
120	198020.1	198583.3	155660.4	88904.2	641168.0
140	209294.2	199779.3	158062.5	126361.0	693496.9
200	288515.9	266009.1	237609.9	163676.3	955811.2
260	364368.8	342441.2	316121.4	270172.4	1293103.8
320	431325.9	410221.5	361473.9	369051.5	1572072.8
400	490630.8	465434.4	465297.7	441242.1	1862605.1
480	566625.7	561515.4	545145.2	516562.2	2189848.5
640	747948.4	740011.5	695780.1	647444.6	2831184.5
800	759702.2	734815.3	778195.6	722163.2	2994876.3
960	811649.3	782186.6	745790.7	706684.4	3046311.0

Table D.105 Byte throughputs for Bryhni-XSIZE-S-1.0 (4-server), $s = 1.143$

Appendix E:

Stem-and-Leaf Plots for Response Times

Frequency	Stem &	Leaf
4211.00	0 .	001111122222233333333334444444444
6300.00	0 .	55555555555666666666677777777777888888888999999999
4811.00	1 .	00000001111111222222223333333444444
3374.00	1 .	5555556666677777888889999
3183.00	2 .	00001112222233333444444
4030.00	2 .	5555566666677777888888999999
3499.00	3 .	00000011111222223333344444
2218.00	3 .	5555666777788899
1651.00	4 .	001112233444
1782.00	4 .	5566677888999
2251.00	5 .	0001111222333444
1409.00	5 .	5566677889
1197.00	6 .	011223344
1214.00	6 .	5566778899
855.00	7 .	01234
685.00	7 .	56789
554.00	8 .	01234
693.00	8 .	56789
605.00	9 .	01234
524.00	9 .	56789
558.00	10 .	01234
476.00	10 .	5679&
453.00	11 .	01234
207.00	11 .	5&
2810.00	Extremes	(>=11.9)

Stem width: 1.00
Each leaf: 137 case(s)

Figure E.1 Stem-and-leaf plot for the response times (XSIZE-G)

Frequency	Stem &	Leaf
277.00	0 .	&
5535.00	0 .	556666777777888888889999999
6836.00	1 .	00000011111112222222333333444444
5394.00	1 .	5555556666667777788888999
3747.00	2 .	000111222333344444
4613.00	2 .	55555566667777888899999
9374.00	3 .	0000001111111112222222222233333333334444444444
4313.00	3 .	555555666677778888999
967.00	4 .	00123&
511.00	4 .	789&
501.00	5 .	01&
620.00	5 .	789&
1747.00	6 .	01223344
2245.00	6 .	55667788899
1272.00	7 .	001234
4870.00	Extremes	(>=7.5)

Stem width: 1.00
Each leaf: 204 case(s)

Figure E.2 Stem-and-leaf plot for the response times (RR-G)

Frequency	Stem &	Leaf
286.00	0 .	&
85.00	0 .	&
132.00	1 .	&
4088.00	1 .	5677888899999
17121.00	2 .	00000111122222222223333333333333333333444444444444444
11686.00	2 .	55555555555566666666667777788899
1056.00	3 .	01&
1090.00	3 .	789&
2555.00	4 .	0123344
5062.00	4 .	5566777888999
3071.00	5 .	000112234
504.00	5 .	&
81.00	6 .	&
75.00	6 .	&
80.00	7 .	&
341.00	7 .	&
17.00	8 .	&
4501.00	Extremes	(>=8.0)

Stem width: 1.00
Each leaf: 373 case(s)

Figure E.3 Stem-and-leaf plot for the response times (CONN-G)

Frequency	Stem &	Leaf
3776.00	0 .	00111111112222222333333444444444
5082.00	0 .	555555555566666666677777777888888899999999
5568.00	1 .	00000000001111111112222222233333333444444444
5397.00	1 .	5555555555666666666777777778888888899999999
4321.00	2 .	00000011111122222233333333444444444
4285.00	2 .	55555555666666666777777788888899999999
3691.00	3 .	000000111111222222333334444444
2601.00	3 .	555555666677777888999
2181.00	4 .	00001111222333444
1748.00	4 .	55566677788899
1493.00	5 .	000111222334
805.00	5 .	56789
1193.00	6 .	001122334
1114.00	6 .	5566778899
953.00	7 .	00112234
782.00	7 .	56789
781.00	8 .	012344
991.00	8 .	556677889
726.00	9 .	01234
457.00	9 .	56789
473.00	10 .	0123&
3085.00	Extremes	(>=10.4)

Stem width: 1.00
Each leaf: 122 case(s)

Figure E.4 Stem-and-leaf plot for the response times (RT-G)

Frequency	Stem &	Leaf
662.00	0 .	013444&
5537.00	0 .	5566666777777777888888888888999999999999
6398.00	1 .	000000000001111111122222222233333333444444444
5335.00	1 .	55555555556666666667777788888888999999
4642.00	2 .	000000111112222223333333344444444
5853.00	2 .	55555566666666677777778888888899999999
5245.00	3 .	000000000001111111122222233333444444
5299.00	3 .	55555566666666677777778888888899999999
2026.00	4 .	000011122233344
995.00	4 .	5566789
683.00	5 .	01234
1001.00	5 .	55667899
1045.00	6 .	0011234
966.00	6 .	5678899
1366.00	7 .	0011223344
1311.00	7 .	5556667788&
4239.00	Extremes	(>=7.9)

Stem width: 1.00
Each leaf: 140 case(s)

Figure E.5 Stem-and-leaf plot for the response times (RAN-G)

Bibliography

- [Almeida 1996] Virgilio Almeida, Azer Bestavros, Mark Crovella and Adriana {deOliveira}" (1996). "Characterizing Reference Locality in the WWW." Proceedings of 1996 International Conference on Parallel and Distributed Information Systems (PDIS '96): 92--103.
<http://citeseer.nj.nec.com/almeida96characterizing.html>
- [AltaVista 2002] AltaVista (2002). <http://www.altavista.com>
- [Anderson 1996] E. Anderson, D. Patterson and E. Brewer (1996). "The Magicrouter: An Application of Fast Packet Interposing." Second Symposium on Operating Systems Design and Implementation.
<http://www.cs.berkeley.edu/~eanders/projects/magicrouter/>
- [Andresen 1996] D. Andresen, T. Yang, O. Egecioglu, O.H. Ibarra and T.R. Smith (1996). Scalability Issues for High Performance Digital Libraries on the World Wide Web. Advances in Digital Libraries: 139-148.
<http://citeseer.nj.nec.com/andresen96scalability.html>
- [Andresen 1998] Daniel Andresen and Timothy McCune (1998). Towards a Hierarchical Scheduling System for Distributed WWW Server Clusters. HPDC: 301-.
<http://citeseer.nj.nec.com/48884.html>

- [Andresen 1996] Daniel Andresen, Tao Yang, Vegard Holmedahl and Oscar H. Ibarra (1996). "SWEB: Towards a Scalable World Wide Web Server on Multicomputers." Proc. of 10th IEEE International Symp. on Parallel Processing (IPPS'96): 850-856. <http://citeseer.nj.nec.com/andresen96swweb.html>
- [Apple Computer 1996] Inc. Apple Computer (1996). AppleTalk Data Stream Protocol (ADSP). <http://developer.apple.com/techpubs/mac/Networking/Networking-98.html>
- [Arlitt 2000] M.F. Arlitt and T. Jin (2000). "A workload characterization study of the 1998 World Cup Web site." IEEE Network **14**(3): 30-37
- [Arlitt 1996] Martin F. Arlitt and Carey L. Williamson (1996). Web Server Workload Characterization: The Search for Invariants. Measurement and Modeling of Computer Systems: 126-137. citeseer.nj.nec.com/arlett96web.html
- [Aversa 2000] Luis Aversa and Azer Bestavros (2000). "Load Balancing a Cluster of Web Servers using Distributed Packet Rewriting." Proceedings of the 2000 IEEE International Performance, Computing, and Communications Conference: 24-29. citeseer.nj.nec.com/aversa00load.html
- [Banga 1999] Gaurav Banga and Peter Druschel (1999). "Measuring the Capacity of a Web Server Under Realistic Loads." World Wide Web **2**(1-2): 69-83. citeseer.nj.nec.com/banga99measuring.html
- [Barford 1998] P. Barford and M. Crovella (1998). "Generating representative workloads for network and server performance evaluation." Proceedings of ACM SIGMETRICS '98 International Conference on Measurement and Modeling of Computer Systems: 151--160. <http://citeseer.nj.nec.com/barford98generating.html>
- [Barford 1999] Paul Barford and Mark Crovella (1999). A performance evaluation of Hyper Text Transfer Protocols. Proc. ACM Sigmetrics, Atlanta. <http://citeseer.nj.nec.com/98399.html>

- [Barford 2001] Paul Robert Barford (2001). Modeling, Measurement And Performance Of World Wide Web Transactions. graduate School of Arts and Sciences. Boston, Boston University: 127. <http://citeseer.nj.nec.com/barford01modeling.html>
- [Berners-Lee 1996] T. Berners-Lee, R. Fielding and H. Nielsen (1996). "Hypertext Transfer Protocol -- HTTP/1.0." RFC 1945. <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1945.html>
- [Bestavros 1998] Azer Bestavros, Mark Crovella, Jun Liu and David Martin (1998). "Distributed Packet Rewriting." Proceedings of ICNP'98:The 6th International Conference on Network Protocols. citeseer.nj.nec.com/509.html
- [Bestavros 1998] Azer Bestavros, Mark Crovella, Jun Liu and David Martin (1998). Distributed Packet Rewriting and its Application to Scalable Server Architectures. Boston, MA, Boston University. <http://citeseer.nj.nec.com/54392.html>
- [Branch 1999] Philip Branch, Greg Egan and Bruce Tonkin (1999). "Modeling Interactive Behavior of a Video based Multimedia System." ACM Sigmetric'99
- [Bray 1996] T. Bray (1996). "Measuring the Web." The World Wide Web Journal 1(3). http://www5conf.inria.fr/fich_html/papers/P9/Overview.html
- [Breslau 2000] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John S. Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu and Haobo Yu (2000). "Advances in Network Simulation." IEEE Computer 33(5): 59-67. <http://www.isi.edu/~johnh/PAPERS/Breslau00a.html>
- [Brisco 1995] T. Brisco (1995). "DNS Support for Load Balancing." Network Working Group RFC 1794
- [Bryhni 2000] Haakon Bryhni, Espen Klovning and Øivind Kure (2000). "A Comparison of Load Balancing Techniques for Scalable Web Servers." IEEE Network July/August: 58-64. <http://www.comsoc.org/ni/private/2000/jul/Bryhni.html>

- [Cardellini 2000] V. Cardellini, M. Colajanni and P. Yu (2000). Geographic load balancing for scalable distributed Web systems. IEEE Mascots 2000, San Francisco, CA. <http://citeseer.nj.nec.com/cardellini00geographic.html>
- [Cardellini 2001] Valeria Cardellini, Emiliano Casalicchio and Michele Colajanni (2001). "A Performance Study of Distributed Architectures for the Quality of Web Services." Proceedings of the Hawai'i International Conference On System Sciences. <http://citeseer.nj.nec.com/376183.html>
- [Cardellini 1999] Valeria Cardellini, Michele Colajanni and Philip S. Yu (1999). "Dynamic Load Balancing on Web-Server Systems." IEEE Internet Computing 3(3): 28-39. <http://www.ece.eng.wayne.edu/~czxu/ece7995/reading/colajanni-ic99.pdf>
- [Cardellini 1999] Valeria Cardellini, Michele Colajanni and Philip S. Yu (1999). "Redirection Algorithms for Load Sharing in Distributed Web-server Systems." Proceedings of the 19th IEEE International Conference on Distributed Computing Systems. <http://www.computer.org/proceedings/icdcs/0222/02220528abs.htm>
- [Carrera 2000] E. Carrera and R. Bianchini (2000). "Evaluating Cluster-Based Network Servers." Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing: 63--70. citeseer.nj.nec.com/carrera00evaluating.html
- [Carrera 1999] Enrique V. Carrera and Ricardo Bianchini (1999). Analytical and Experimental Evaluation of Cluster-Based Network Servers, Department of Computer Science, University of Rochester. <http://citeseer.nj.nec.com/article/carrera99analytical.html>
- [Carrera 2000] Enrique V. Carrera and Ricardo Bianchini (2000). Efficiency vs. Portability in Cluster-Based Network Servers, department of Computer Science, Rutgers University. <http://citeseer.nj.nec.com/452791.html>

- [Carter 1996] Robert Carter and Mark Crovella (1996). Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks, Computer Science Department, Boston University. <http://citeseer.nj.nec.com/carter96dynamic.html>
- [Catledge 1995] Lara D. Catledge and James E. Pitkow (1995). "Characterizing browsing strategies in the World-Wide Web." Computer Networks and ISDN Systems **27**(6): 1065--1073. <http://citeseer.nj.nec.com/catledge95characterizing.html>
- [Chankhunthod 1996] Anawat Chankhunthod, Peter Danzig, Chuck Neerdaels, Michael F. Schwartz and Kurt J. Worrell (1996). A Hierarchical Internet Object Cache. Proceedings of the 1996 {Usenix} Technical Conference, San Diego, CA. <http://citeseer.nj.nec.com/135155.html>
- [Chen 1999] Yu Chen (1999). QOS prediction and evaluation for Networked Telelearning Applications. School of engineering. Burnaby, Simon Fraser University: 133
- [Cisco Systems 2001] Inc. Cisco Systems (2001). Cisco LocalDirector Configuration and Command Reference Guide. <http://www.cisco.com/univercd/cc/td/doc/product/iaabu/localdir/ldv42/421guide/index.htm>
- [Cisio 1997] Cisio (1997). The effects of distributing load randomly to servers (White Paper). http://www.cisco.com/warp/public/cc/pd/cxsr/dd/tech/ddran_wp.pdf
- [Cluts 1998] Nancy Winnick Cluts (1998). Using WCAT to Stress-Test IIS. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dniis/html/usingwcat.asp>
- [Colajanni 1998] Michele Colajanni, Philip S. Yu and Valeria Cardellini (1998). "Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers." International Conference on Distributed Computing Systems: 295-302. citeseer.nj.nec.com/colajanni98dynamic.html

- [Coll 2000] David C. Coll (2000). Trends in Telecommunications and Their Impact on TeleLearning, the TeleLearning Network of Centers of Excellence:
<http://www.sce.carleton.ca/faculty/coll/TLNReportbody.htm>.
<http://www.sce.carleton.ca/faculty/coll/TLNReportbody.htm>
- [Corporation 1996] Standard Performance Evaluation Corporation (1996). An Explanation of the SPECweb96 Benchmark.
<http://open.specbench.org/osg/web96/webpaper.html>
- [Corporation 2000] Standard Performance Evaluation Corporation (2000). SPECWeb96 Benchmark. <http://www.specbench.org/osg/web96/>
- [Corporation 2001] Standard Performance Evaluation Corporation (2001). SPECweb99 Benchmark. <http://www.spec.org/osg/web99>
- [Courage 1998] M. Courage and S. Manley (1998). “An Evaluation of CGI Traffic and Its Effect on WWW Latency.” <http://www.eecs>.
<http://citeseer.nj.nec.com/286607.html>
- [Crovella 1995] Mark Crovella and Azer Bestavros (1995). “Explaining World Wide Web Traffic Self-Similarity.” Technical Report BUCS-TR-95-015.
<http://citeseer.nj.nec.com/crovella95explaining.html>
- [Crovella 1997] Mark E. Crovella and Azer Bestavros (1997). “Self-similarity in World Wide Web traffic: evidence and possible causes.” IEEE /ACM Transactions on Networking **5**(6): 835-846. <http://citeseer.nj.nec.com/crovella96selfsimilarity.html>
- [Cunha 1995] Carlos Cunha, Azer Bestavros and Mark Crovella (1995). Characteristics of WWW Client-based Traces, Boston University, Computer Science Department. <http://citeseer.nj.nec.com/44710.html>
- [Cunha 1995] Carlos Cunha, Azer Bestavros and Mark Crovella (1995). “Characteristics of WWW Client-based Traces.” Technical Report TR-95-010.
<http://citeseer.nj.nec.com/cunha95characteristics.html>

- [Dahlin 1998] A. Dahlin, M. Froberg, J. Walerud and P. Winroth (1998). EDDIE: A Robust and Scalable Internet Server.
<http://www.dcc.ufmg.br/~meira/www/dahlin.pdf>
- [Damani 1997] Om P. Damani, P. Emerald Chung and Yennun Huang (1997). "ONE-IP: Techniques for Hosting a Service on a Cluster of Machines." Computer Networks and ISDN Systems **29**(8--13): 1019--1027.
<http://www.scope.gmd.de/info/www6/technical/paper196/paper196.html>
- [Davison 2001] Brian D. Davison (2001). HTTP Simulator Validation Using Real Measurements: A Case Study. Proceedings of MASCOTS 2001.
<http://citeseer.nj.nec.com/davison01http.html>
- [Delgadillo 2000] Kevin Delgadillo (2000). White Paper: Cisco DistributedDirector.
http://www.cisco.com/warp/public/cc/pd/cxsr/dd/tech/dd_wp.htm
- [Deng 1996] S. Deng (1996). "Empirical model of WWW document arrivals at access link." Proceedings of the 1996 IEEE International Conference on Communication. <http://citeseer.nj.nec.com/deng96empirical.html>
- [Dykes 2000] Sandra G. Dykes, Kay A. Robbins and Clinton L. Jeffery (2000). "An Empirical Evaluation of Client-Side Server Selection Algorithms." Proc. of IEEE Infocom: 1361-1370. citeseer.nj.nec.com/dykes00empirical.html
- [Egevang 1994] K. Egevang and P. Francis (1994). The IP Network Address Translator (NAT), RFC 1631. <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1631.html>
- [Engelschall 1998] Ralf S. Engelschall (1998). Load Balancing Your Web Site: Practical Approaches for Distributing HTTP Traffic. Web Techniques Magazine. **3**.
<http://www.webtechniques.com/archives/1998/05/engelschall/>
- [Erramilli 1996] Ashok Erramilli, Onuttom Narayan and Walter Willinger (1996). "Experimental queueing analysis with long-range dependent packet traffic." IEEE /ACM Transactions on Networking **4**(2): 209--223.
<http://citeseer.nj.nec.com/erramilli96experimental.html>

- [Fall 2002] Kevin Fall and Kannan Varadhan (2002). The ns Manual.
<http://www.isi.edu/nsnam/ns/ns-documentation.html>
- [Fielding 1997] R. Fielding, J. Mogul, J. Gettys, H. Frystyk and T. Berners-Lee (1997).
“Hypertext Transfer Protocol -- HTTP/1.1.” RFC 2068. <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2068.html>
- [Floyd 2001] Sally Floyd and Vern Paxson (2001). “Difficulties in Simulating the Internet.” IEEE/ACM Transactions on Networking.
<http://citeseer.nj.nec.com/floyd01difficulties.html>
- [Force 2002] The Internet Engineering Task Force (2002). <http://www.ietf.org/>
- [Foundation 2002] The Apache Software Foundation (2002).
<http://httpd.apache.org/docs/>
- [Fox 1997] A. Fox (1997). Extensible Cluster-Based Scalable Network Services. Proc. 16th ACM Symp. Operating System Principles (SOSP-16), St. Malo, France.
<http://citeseer.nj.nec.com/fox97extensible.html>
- [Fox 1997] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer and Paul Gauthier (1997). “Cluster-Based Scalable Network Services.” Symposium on Operating Systems Principles: 78-91. citeseer.nj.nec.com/fox97clusterbased.html
- [Freeman 1995] R Freeman (1995). Practical Data Communications, John Wiley and Sons.
- [Gage 2000] Chris Gage (2000). White paper, IBM Network Dispatcher Version 3.0
- [Gan 2000] Xuehong Gan, Trevor Schroeder, Steve Goddard and Byrav Ramamurthy (2000). “LSMAC vs. LSNAT: Scalable Cluster-based Web Servers.” Cluster Computing 3(3): 175-185. <http://www.baltzer.nl/oasis.htm/327470>
- [Glassman 1994] S. Glassman (1994). “A caching relay for the World Wide Web.” Computer Networks and ISDN Systems 27(2).
<http://www1.cern.ch/PapersWWW94/steveg.ps>

- [Group 2000] The NSS Group (2000). Cisco CSS 11800 Content Services Switch —An NSS Group Report.
http://www.cisco.com/warp/public/cc/pd/si/11000/prodlit/cstes_wi.htm
- [Harbaugh 1999] Logan Harbaugh (1999). A Delicate Balance.
<http://www.informationweek.com/718/18olbal.htm>
- [Heidemann 2001] John Heidemann, Kevin Mills and Sri Kumar (2001). “Expanding confidence in network simulation.” IEEE Network **15**(5): 58-63.
<http://www.isi.edu/~johnh/PAPERS/Heidemann00c.html>
- [Helmy 1997] Ahmed Helmy and Satish Kumar (1997). VINT: Virtual InterNetwork Testbed. <http://www.isi.edu/nsnam/vint/>
- [Hu 1998] James C. Hu, Sumedh Mungee and Douglas C. Schmidt (1998). JAWS: Understanding High Performance Web Systems.
<http://www.cs.wustl.edu/~jxh/research/research.html>
- [Huberman 1998] Bernardo A. Huberman, Peter L. T. Pirolli, James E. Pitkow and Rajan M. Lukose (1998). “Strong Regularities in World Wide Web Surfing.” Science **280**(5360): 95-97. <http://www.sciencemag.org/cgi/content/abstract/280/5360/95>
- [IEEE. 2000] IEEE. (2000). IEEE 802.5 Web Site. <http://www.8025.org/>
- [Ingham 2000] David B. Ingham and Santosh K. Shrivastava (2000). “Constructing Dependable Web Service.” IEEE Internet Computing **4**(1): 25-33
- [INKTOMI 2002] INKTOMI (2002). <http://www.inktomi.com/>
- [Iyengar 1997] A. Iyengar, E. MacNair and T. Nguyen (1997). “An Analysis of Web Server Performance.” Proceedings of GLOBECOM '97.
<http://citeseer.nj.nec.com/83344.html>
- [Iyengar 1997] Arun Iyengar and Jim Challenger (1997). Improving Web Server Performance by Caching Dynamic Data. {USENIX} Symposium on Internet Technologies and Systems. <http://citeseer.nj.nec.com/iyengar97improving.html>

- [Iyengar 1999] Arun K. Iyengar, Mark S. Squillante and Li Zhang (1999). "Analysis and Characterization of Large-Scale Web Server Access Patterns and Performance." World Wide Web. <http://citeseer.nj.nec.com/iyengar99analysis.html>
- [Jacobson 1988] V. Jacobson (1988). "Congestion Avoidance and Control." ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988 **18**(4): 314--329.
<http://citeseer.nj.nec.com/jacobson88congestion.html>
- [Katz 1994] Eric Dean Katz, Michelle Butler and Robert McGrath (1994). "A Scalable HTTP Server: The NCSA Prototype." Computer Networks and ISDN Systems **27**: 155-164.
<http://archive.ncsa.uiuc.edu/InformationServers/Conferences/CERNwww94/www94.ncsa.html>
- [Laboratory 2001] UCLA Parallel Computing Laboratory (2001). Parsec: Parallel Simulation Environment for Complex Systems.
<http://pcl.cs.ucla.edu/projects/parsec/>
- [Ladkin 2000] Peter Ladkin (2000). Web Server Performance. <http://nakula.rvs.uni-bielefeld.de/made/artikel/Web-Bench/web-bench.html>
- [Leland 1993] Will E. Leland, Murad S. Taqq, Walter Willinger and Daniel V. Wilson (1993). On the self-similar nature of Ethernet traffic. ACM SIGCOMM. D. P. Sidhu. San Francisco, California: 183--193.
<http://citeseer.nj.nec.com/leland93selfsimilar.html>
- [Levy-Abegnoli 1999] Eric Levy-Abegnoli, Arun Iyengar, Junehwa Song and Daniel Dias (1999). "Design and Performance of a Web Server Accelerator." IEEE INFOCOM '99. <http://citeseer.nj.nec.com/150491.html>
- [Liu 1999] zhen Liu, Nocolas Niclausse, Cesar Jalpa-Villanueva and Sylvain Barbier (1999). Traffic Model and Performance Evaluation of Web Servers.
<http://www.inria.fr/rrrt/rr-3840.html>

- [Mah 1997] B. Mah (1997). "An Empirical Model of HTTP Network Traffic." Proc. InfoComm '97. <http://citeseer.nj.nec.com/mah97empirical.html>
- [Manley 1997] Stephen Manley and Margo Seltzer (1997). "Web facts and Fantasy." Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems. <http://citeseer.nj.nec.com/manley97web.html>
- [Manley 1998] Stephen Manley, Margo Seltzer and Michael Courage (1998). "A Self-Scaling and Self-Configuring Benchmark for Web Servers." ACM-SIGMETRICS '98: 270-271. citeseer.nj.nec.com/83040.html
- [Marc Greis 2002] VINT group Marc Greis (2002). Tutorial for the Network Simulator ns. <http://www.isi.edu/nsnam/ns/tutorial/index.html>
- [McGrath 1996] Robert E. McGrath (1996). Measuring the Performance of HTTP Daemons.
<http://archive.ncsa.uiuc.edu/InformationServers/Performance/Benchmarking/bench.html>
- [McGrath 1996] Robert E. McGrath (1996). Performance of Several Web Server Platforms.
<http://archive.ncsa.uiuc.edu/InformationServers/Performance/Platforms/report.html>
- [Microsoft 1999] Microsoft (1999). The Microsoft® Web Capacity Analysis Tool (WCAT).
http://www.microsoft.com/ISN/downloads/wcat__a_tool_to_test_929.asp
- [Microsystems 2002] SUN Microsystems (2002). The source for Java Technology.
<http://www.javasoft.com/>
- [Mindcraft 2002] Inc. Mindcraft (2002). WebStone:The Benchmark for Web Servers.
<http://www.mindcraft.com/webstone/>
- [Mogul 1995] Jeffrey C. Mogul (1995). "The Case for Persistent-Connection HTTP." Proceedings of the SIGCOMM '95 Conference on Communications Architectures

- and Protocols: 299-313.
- <http://www.research.compaq.com/wrl/techreports/abstracts/95.4.html>
- [Mogul 1995] Jeffrey C. Mogul (1995). Network behavior of a busy web server and it's clients. Palo Alto, CA, DEC Western research laboratory.
- <ftp://gatekeeper.dec.com/pub/DEC/WRL/research-reports/WRL-TR-95.5.ps>
- [Molina 2000] Maurizio Molina, Paolo Castelli, Gianluca Foddìs and CSELT (2000). “Web Traffic modeling, Exploiting TCP Connections' Temporal Clustering through HTML-REDUCE.” IEEE Network **May/June**: 46-55
- [Mosberger 1998] David Mosberger and Tai Jin (1998). httpperf---A Tool for Measuring Web Server Performance.
- http://www.hpl.hp.com/personal/David_Mosberger/httpperf.html
- [Network 2002] SSF Research Network (2002). Scalable Simulation Framework (SSF).
- <http://www.ssfnet.org/homePage.html>
- [Networks 2002] Nortel Networks (2002). Alteon Web Switching Portfolio.
- <http://www.nortelnetworks.com/products/01/alteon/index.html>
- [Nielsen 1997] H. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie and C. Lilley (1997). “Network performance effects of HTTP/1.1, CSS1, and PNG.” Proceedings of ACM SIGCOMM'97 Conference.
- <http://www.w3.org/Protocols/HTTP/Performance/Pipeline.html>
- [OPNET Technologies 2002] Inc. OPNET Technologies (2002). Optimum Network Performance. <http://www.opnet.com>
- [Pai 1998] Vivek S. Pai, Mohit Aron, Gaurov Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel and Erich Nahum (1998). Locality-aware request distribution in cluster-based network servers. Proceedings of the 8th international conference on Architectural support for programming languages and operating systems, San Jose, CA USA. <http://citeseer.nj.nec.com/pai98localityaware.html>

- [Park 1996] Kihong Park, Gitae Kim and Mark Crovella (1996). On the relationship between file sizes, transport protocols, and self-similar network traffic. Proc. IEEE International Conference on Network Protocols (ICNP'96).
<http://citeseer.nj.nec.com/park96relationship.html>
- [Paxson 1994] Vern Paxson (1994). "Empirically derived analytic models of wide-area TCP connections." IEEE\ACM Transactions on Networking 2(4): 316--336.
<http://citeseer.nj.nec.com/paxson93empiricallyderived.html>
- [Perkins 1996] C. Perkins (1996). IP Encapsulation within IP, RFC 2003.
<http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2003.html>
- [Phifer 1999] Lisa Phifer (1999). White paper. <http://www.coyotepoint.com/cpwp.pdf>
- [Pirolli 1999] Peter Pirolli and James E. Pitkow (1999). "Distributions of Surfers' Paths Through the World Wide Web: Empirical Characterizations." World Wide Web 2(1-2): 29-45. <http://www.parc.xerox.com/istl/projects/uir/pubs/pdf/UIR-R-1999-02-Pirolli-WebJournal-Paths.pdf>
- [Pitkow 1999] James E. Pitkow (1999). "Summary of WWW characterizations." World Wide Web 2(1-2): 3-13. <http://citeseer.nj.nec.com/james99summary.html>
- [Postel 1980] J.B. Postel (1980). User Datagram Protocol.
<http://www.faqs.org/rfcs/rfc768.html>
- [Postel 1981] Jon B. Postel (1981). "Transmission Control protocol." RFC 793, Network Information Center, SRI International
- [Radware 2002] Inc. Radware (2002). Web Server Director (WSD).
<http://www.radware.com/content/products/wsd.asp>
- [Rubarth-Lay 1996] James Rubarth-Lay (1996). "Keeping the 400lb. Gorilla at Bay: Optimizing Web Performance."
<http://eunuch.ddg.com/LIS/CyberHornsS96/j.rubarth-lay/PAPER.html>.
<http://eunuch.ddg.com/LIS/CyberHornsS96/j.rubarth-lay/PAPER.html>

- [Rubenstein 1984] Richard Rubenstein, Happy M. Hersh and Henry F. Ledgard (1984).
The Human Factor: Designing Computer Systems for People. Burlington, MA,
 Digital Press
- [Schroeder 2000] Trevor Schroeder, Steve Goddard and Byrav Ramamurthy (2000).
 “Scalable Web Server Clustering Technologies.” IEEE Network **May/June** : 38-
 45. <http://www.ece.eng.wayne.edu/~czxu/ece7995/reading/ServerReview-Schroeder.pdf>
- [Seltzer 1995] M. Seltzer and J. Gwertzman (1995). “The Case for Geographical Push-caching.” Proceedings of the 1995 Workshop on Hot Operating Systems.
<http://citeseer.nj.nec.com/seltzer95case.html>
- [Slothouber 1996] L. Slothouber (1996). “A Model of Web Server Performance.” Proc. of 5 th Conference On WWW.
<http://www.geocities.com/webserverperformance/modelpaper.html>
- [Sneed 2000] Harry Sneed (2000). “Testing Software for Internet Applications.”
Software Focus **1**(1): 15-22
- [Song 1998] Junehwa Song, Eric Levy- Abegnoli, Arun Iyengar and Daniel Dias (1998).
A Scalable and Highly Available Web Server Accelerator. Proceedings of
 ACM/IEEE Supercomputing '98 (SC98), Orlando, Florida.
<http://citeseer.nj.nec.com/372060.html>
- [Sound 2000] Puget Sound (2000). Higher learning goes the distance. ComputerUser: 18-
 22
- [Speight 1999] Evan Speight, Hazim Abdel-Shafi and John K. Bennett (1999). “Realizing the performance potential of the virtual interface architecture.” International Conference on Supercomputing: 184-192.
citeseer.nj.nec.com/speight99realizing.html

- [Squillante 1999] M.S. Squillante, D.D. Yao and L.Zhang (1999). Internet traffic: Periodicity, tail behavior and performance implications. System Performance Evaluation: Methodologies and Applications. E. Gelenbe, CRC Press
- [Squillante 1999] M.S. Squillante, D.D. Yao and L.Zhang (1999). “Web traffic modeling and server performance analysis.” Proceedings of the IEEE Conference on Decision and Control (CDC)
- [Srisuresh 1998] P. Srisuresh and D. Gan (1998). Load Sharing using IP Network Address Translation (LSNAT), RFC2391. <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2391.html>
- [Tanenbaum 1996] A Tanenbaum (1996). Computer Networks, Prentice Hall
- [Tauscher 1997] L. Tauscher and S. Greenberg (1997). “How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems.” International Journal of Human Computer Studies, Special issue on World Wide Web Usability **47**(1): 97-138.
<http://www.cpsc.ucalgary.ca/Redirect/grouplab/papers/1997/97-RevisitationPatterns.CHI/sg.html>
- [Team 1998] NCSA HTTPd Development Team (1998). The CGI Specification.
<http://hoohoo.ncsa.uiuc.edu/cgi/interface.html>
- [(TL•NCE) 2002] The TeleLearning Network of Centres of Excellence (TL•NCE) (2002). <http://www.telelearn.ca/>
- [Trent 1995] G. Trent and M. Sake (1995). “WebSTONE: The First Generation in HTTP Server Benchmarking.”
<http://www.sgi.com/Products/WebFORCE/WebStone/paper.html>.
<http://citeseer.nj.nec.com/trent95webstone.html>
- [W3C 1999] W3C (1999). Hypertext Transfer Protocol -- HTTP/1.1.
<http://www.w3.org/Protocols/rfc2616/rfc2616.html>

- [W3C 2001] W3C (2001). Final HTTP-NG Activity Statement.
<http://www.w3.org/Protocols/HTTP-NG/Activity.html>
- [W3C 2001] W3C (2001). HTTP - Hypertext Transfer Protocol.
<http://www.w3.org/Protocols/>
- [W3C 2002] W3C (2002). HyperText Markup Language. <http://www.w3.org/MarkUp/>
- [Wang 1998] Zhe Wang and Pei Cao (1998). Persistent Connection Behavior of Popular Browsers. <http://www.cs.wisc.edu/~cao/papers/persistent-connection.html>
- [Willinger 1997] Walter Willinger, Murad S. Taqqu, Robert Sherman and Daniel V. Wilson (1997). "Self-similarity through high-variability: statistical analysis of Ethernet (LAN) traffic at the source level." IEEE\ACM Transactions on Networking **5**(1): 71--86. <http://citeseer.nj.nec.com/willinger97selfsimilarity.html>
- [Yoshikawa 1997] Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson and David Culler (1997). Using Smart Clients to Build Scalable Services. Proceedings of the USENIX 1997 Annual Technical Conference, Anaheim, California.
http://www.usenix.org/publications/library/proceedings/ana97/full_papers/yoshikawa/yoshikawa_html/usenix97.html or
<http://citeseer.nj.nec.com/yoshikawa97using.html>
- [Zeus 2002] Zeus (2002). Zeus Load Balancer. <http://www.zeus.com/products/zlb/>
- [Zhang 1999] Wensong Zhang, Shiyao Jin and Quanyuan Wu (1999). Creating Linux Virtual Servers. LinuxExpo 1999 Conference, Linux Virtual Server Project.
<http://www.linuxvirtualserver.org/linuxexpo.html>
- [Zhu 1998] Huican Zhu, Ben Smith and Tao Yang (1998). A Scheduling Framework for Web Server Clusters with Intensive Dynamic Content Processing.
<http://citeseer.nj.nec.com/zhu98scheduling.html>

[Zhu 1999] Huican Zhu, Ben Smith and Tao Yang (1999). “Hierarchical Resource Management for Web Server Clusters with Dynamic Content.” ACM Sigmetrics'99: 198-199