Georgia State University
Digital Archive @ GSU

Computer Science Theses | Department of Computer Science

12-5-2006

Scalable Proxy Architecture for Mobile and Peer-to-Peer Networks

Praveena Jayanthi

Follow this and additional works at: http://digitalarchive.gsu.edu/cs_theses

Recommended Citation
Jayanthi, Praveena, "Scalable Proxy Architecture for Mobile and Peer-to-Peer Networks" (2006). *Computer Science Theses*. Paper 34.

This Thesis is brought to you for free and open access by the Department of Computer Science at Digital Archive @ GSU. It has been accepted for inclusion in Computer Science Theses by an authorized administrator of Digital Archive @ GSU. For more information, please contact digitalarchive@gsu.edu.

Scalable Proxy Architecture for Mobile and Peer-to-Peer Networks

by

Praveena Jayanthi

Under the Direction of Sushil k Prasad

ABSTRACT

The growth of wireless telecommunications has stipulated the interest for anywhere-anytime computing. The synergy between networking and mobility will engender new collaborative applications with mobile devices on heterogeneous platforms. One such middleware is "SYSTEM ON MOBILE DEVICES", SYD developed by the Yamacraw Embedded Systems research team. This type of middleware is an opening step towards Peer-to-Peer mobile networks. This project envisioned collaborative applications among mobile devices and PDAs were used as servers. This thesis studies various existing architectures in mobile computing and their scalability issues. We also proposed new scalable flexible thick client proxy system FTCPS, an architecture suitable for mobile Peer-to-Peer networks. Our empirical study showed that FTCPS has low response time compared to other architectures.

INDEX WORDS: Proxy, Client, Server, Mobile, Peer-to-Peer Networks, Architecture, Scalable, FTCPS

-

SCALABLE PROXY ARCHITECTURE FOR MOBILE AND PEER-TO-PEER NETWORKS

by

Praveena Jayanthi

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

In the College of Arts and Science

Georgia State University

2006

-

Copyright by

Praveena Jayanthi

2006

-

SCALABLE PROXY ARCHITECTURE FOR MOBILE AND PEER-TO-PEER NETWORKS

by

PRAVEENA JAYANTHI

Major Professor: Sushil K Prasad
Committee: Anu G Bourgeois
Raj Sunderraman

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2006

-

TABLE OF CONTENTS

1. INTRODUCTION....1
1.1 Purpose of examining proxies in computing....1
1.2 Thesis Road-Map....2
2. Proxies in various computing paradigms....3
2.1 Proxy and WWW....3
2.2 Proxy and Mobile Computing....4
2.3 Proxy and Peer-to-Peer Networks....5
2.4 Proxy and Location Based Services....6
2.5 Proxy and Grid Computing....7
2.6 Proxy and Multimedia Applications....8
2.7 Proxy and Middleware....8
2.8 Proxy as a Service....8
2.9 Advantages of using Proxies....9
2.10 Disadvantages of using Proxies....10
2.11 Various Mathematical Distributions and their Implications for evaluating proxies....10
3. Existing architectures for Mobile Computing....12
3.1 Thin-Client Architecture (Proxy as an agent for Mobile Unit)....12
3.2 Full Client Architecture (No way to handle disconnections)....14
3.3 Client-Server with Proxy 15
3.4 Hybrid Client-Server with Proxy 17
3.5 Flexible Thick Client Proxy System (Our Proposal)....20
4. SIMJAVA: Design choice for modeling the simulation....28
4.1 What is SIMJAVA?....28
4.2 The SIMJAVA Design....29
4.3 Overview of SIMJAVA....29
4.3.1 Setting up the simulation 30
4.3.3 What is the trace of a simulation?....30
4.3.4 Sampling distributions....31
4.3.5 Adding statistical measurements....31
4.3.6 What is a transient condition?....33
4.3.6 Defining a transient condition....33
4.3.7 Defining a termination condition....34
5. Experimental set up FOR SIMULATION AND RESULTS....36
5.1 Thin Client Architecture....36
5.2 Full-Client Architecture....40
5.3 Client Proxy Server System....41
5.4 Flexible Thick Client Proxy Server....43
6. Final Results and Concluding Remarks....48
6.1 Concluding Remarks 60
6.2. Future Work....60
7. References....62

-

LIST OF TABLES

Table 1: Trace Statistics.....37
Table 2: Residence Time of Client in Thin Client Architecture.....38
Table 3: Residence Time of Server in Thin Client Architecture.....39
Table 4: Residence Time in Full Client Architecture.....40
Table 5: Residence Time of Proxy in Client Proxy Server System.....41
Table 6: Residence Time of Client in Client Proxy Server System.....42
Table 7: Residence Time of Server in Client Proxy Server System.....43
Table 8: Data Structure(HashMap) for the links in the network cluster.....44
Table 9: Residence Time of Client in FTCPS.....44
Table 10: Residence Time of Proxy in FTCPS.....45
Table 11: Residence Time of Server in FTCPS.....46

-

LIST OF FIGURES

Figure 1: Typical Web Proxy action sequence....3
Figure 2: Proxy and Mobile Computing....4
Figure 3: Proxy Platform....5
Figure 4: Proxy and Peer-to-Peer Networks....6
Figure 5: Proxy and grid computing....7
Figure 6: Proxy Design Pattern....8
Figure 7: Thin-Client Architecture....12
Figure 8: Sequence Diagram of Thin-Client Architecture....13
Figure 9: Full Client Architecture....14
Figure 10: Message Sequence in Full Client Architecture....15
Figure 11: Client-Server with Proxy....15
Figure 12: Message Sequence in Client Proxy Server System....16
Figure 13: Message Sequence in Client Proxy System....17
Figure 14: Hybrid Client-Server with Proxy....18
Figure 15: Message Sequence in Hybrid Client Proxy Server System....19
Figure 16: Message Sequence in Hybrid Client Proxy Server System....20
Figure 17: SyD Architecture....21
Figure 18: Flexible Thick Client Proxy System....22
Figure 19: Message Sequence in FTCPS when MU1 and MU2 are both connected....23
Figure 20: Message Sequence in FTCPS when MU2 is disconnected....24
Figure 21: Message Sequence in FTCPS when MU1 is disconnected after making the request to MU2....25
Figure 22: Message Sequence when MU1 is disconnected after making the request to MU2 which is disconnected....26
Figure 23: Residence Time of a Proxy in Thin Client Architecture....38
Figure 24: Residence Time of a Client in Thin Client Architecture....39
Figure 25: Residence Time of a Server in Thin Client Architecture....39
Figure 26: Residence Time of 100 - 500 Nodes....41
Figure 27: Residence Time of Proxy in a Client Proxy Server System....42
Figure 28: Residence Time of Client in Client Proxy Server System....42
Figure 29: Residence Time Server 100 - 500 Nodes....43
Figure 30: FTCPS Client Residence Time 100 – 500 Nodes....45
Figure 31: FTCPS Proxy Residence Time 100 - 500 Nodes....46
Figure 32: FTCPS Server Residence Time 100 – 500 Nodes....47
Figure 33: Residence Time of Client for 30% disconnection probability....48
Figure 34: Residence Time of Client for 50% disconnection probability....49
Figure 35: Residence Time of Client for 70% disconnection probability....50
Figure 36: Residence Time of Proxy for 30% disconnection probability....51
Figure 37: Residence Time of Proxy for 50% disconnection probability....52
Figure 38: Residence Time of Proxy for 70% disconnection probability....53
Figure 39: Residence Time of Server for 30% disconnection probability....54
Figure 40: Residence Time of Server for 50% disconnection probability....55
Figure 41: Residence Time of Server for 70% disconnection probability....56
Figure 42: Response Time across all architectures at 30% disconnection probability....57
Figure 43: Response Time across all architectures at 50% disconnection probability....58
Figure 44: Response Time across all architectures at 70% disconnection probability....59

-

LIST OF ABBREVIATIONS

1. P2P – Peer to Peer Networks
2. SYD- System of Mobile Devices
3. WWW – World Wide Web
4. FTCPS – Flexible Thick Client Proxy System
5. PVRD - Proxy Viewpoints Model-based Resources Discovery
6. SRS – Set of Requirements Specifications
7. RMI - Remote Method Invocation
8. RMIC - Remote Method Invocation Compiler

1. INTRODUCTION

Proxy is "substitution". The concept of proxy has spawned in to many areas of computing especially WWW and mobile computing. Frequent disconnections and resource dearth are the challenges faced by the wireless world. A proxy can play the role of FILLER between the wireless and wired worlds. The synergy between networking and mobility will engender new collaborative applications with mobile devices on heterogeneous platforms. One such middleware is "SYSTEM ON MOBILE DEVICES", SYD developed by the Yamacraw Embedded Systems research team. SYD is a new paradigm for collaborative application development. SyD consists of a combination of middleware and clientware technologies that enables application development independent of the specific nature of the device, data, services, or network (device locations) that the application is targeted to run over. SyD enables such independent application development through well defined wrapping standards for devices, data, and services. This thesis aims to evaluate the role of proxies in various computing paradigms, a retrospection followed by a vision to design a proxy architecture which is scalable and tolerant to disconnections for SYD best suited for developing extensive collaborative applications.

1.1 Purpose of examining proxies in computing

Proxies have been extensively used as an intermediate computational agent between a client and the server. They also contributed for the increase in scalability and fault tolerance of a system. Paradoxically, the more the functionality incorporated into a proxy, the less scalable the system would become. The complexity and the bottleneck issues are now transferred from the servers to the proxies. So in light of answering the following questions, we would like to examine the role of proxies in various computing paradigms. 1) How to avoid performance degradation? 2) How to increase scalability and fault tolerance of the system? 3) How to increase the level of transparency to disconnections? This survey would be followed by a study whose methodical stance is on tangible quantitative methods for the evaluation of existing architectures. We would also like to propose a new architecture which leverages the benefits of Mobile, WWW and Peer-to-Peer computing. Our approach is motivated by the goal of offloading the functionality of proxies and servers to thick clients.

-

1.2 Thesis Road-Map

We begin Section 1 by a comprehensive examination of the role of proxies in various computing paradigms. Then a detailed analysis of functions being offloaded to proxies is presented. We discuss the relative advantages and disadvantages of using proxies. We then present existing architectures available in mobile and Peer-to-Peer networks followed by an empirical study that compares those with our proposed architecture in residence time and response time metrics. The final section would summarize our findings and present our direction of future research in this area.

2. PROXIES IN VARIOUS COMPUTING PARADIGMS

This chapter details the role of proxies in various computing paradigms like, the World Wide Web, mobile computing, Peer-to-Peer networks, grid computing etc

2.1 Proxy and WWW

A proxy server, sitting in the middle-tier in the WWW infrastructure, serves many web clients and covers a wide scale of web domains consisted of heterogeneous web-sites.

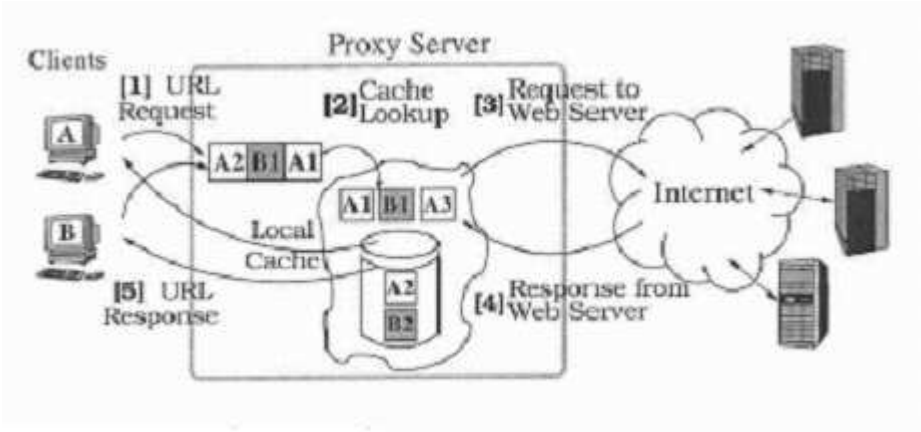


Figure 1: Typical Web Proxy action sequence

Typical Web Proxy action Sequence 1: Figure borrowed from [17]

WBI (Web Browser Intelligence) is a proxy that intercepts the HTTP stream and alters the data thus providing opportunities for personalizing the web experience [2]. This paper also discusses how a confederation of agents can enable the collaboration among web users and improving their web experience. Web server based studies are fewer in number compared to proxy-based studies. Web proxy servers have been used for the web work load characterization so far. Proxy logs have been extensively used to study the effectiveness of caching at proxies and cooperative caching. Markatos et al proposed a web conscious storage management for the web proxy servers that exploit the unique reference characteristics of web page accesses in order to overcome the file I/O limitations [17]. An effective caching policy at the proxies can considerably accelerate the web browsing in wireless customer premises networks. Hadjiefthymiades et al proposed a new

path prediction algorithm to facilitate the dynamic cache relocation to the most probable cells following the roaming users [9]. The above techniques accelerate the web-browsing experience by minimizing the processing times and latencies at the proxies. Lou et al proposed a proxy-based prediction (PPS) for an efficient prediction of web accesses on proxy servers [16]. Their PPS applies a new prediction scheme which employs a two-layer navigation model to capture both inter-site and intrasite access patterns, incorporated with a bottom up prediction mechanism that exploits the locality of reference in the logs of proxy servers. Rabinovich et al observed that sometimes it is better to fetch a web page from a server rather than a peer-proxy [20]. The result of the study was the proposed co-operative caching scheme over wide area networks.

2.2 Proxy and Mobile Computing

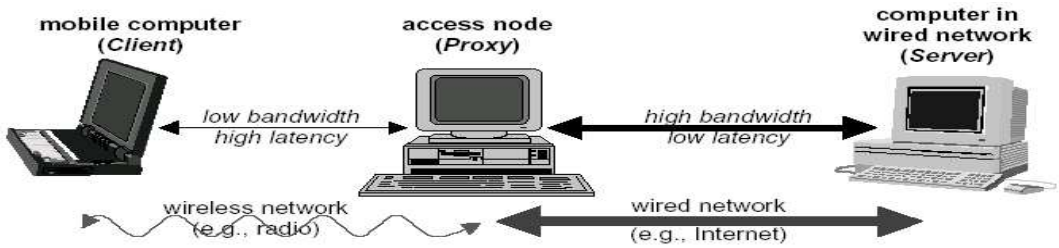


Figure 2: Proxy and Mobile Computing

Communication, mobility and portability are the major issues of Mobile Computing. The wireless world is often termed Long-Thin world due to the longer latencies and thin bandwidth available. Frequent disconnections are a major challenge faced by mobile computing. Automated hoarding techniques are used to give the mobile user an illusion of connection [14]. Besides the automated hoarding techniques, a detailed study is done on the CODA file system to allow the users access files under weakly connected modes [18]. Mobile Computing platforms typically provide a QoS management architecture that facilitates adaptation through the installation of proxies. Predetermined or preconfigured QoS events could trigger the instantiation of filtering, transcoding or caching components into the communication path

between the clients and the servers. Rao et al Propose a Proxy-based platform that implements three key abstractions namely, infolet, applet and devlet. IMobile architecture tries to hide the complexity due to the multiplicity of devices and information sources [21]. The infolet, applet and devlets interact with each other through the LET engine. Devlet is a driver attached to a proxy that sends and receives information using a particular protocol. Infolet tries to give an abstract view of the information space. The let engine supports user and device profiles for personalization and transcoding. The proxy platform is shown in the figure 4.

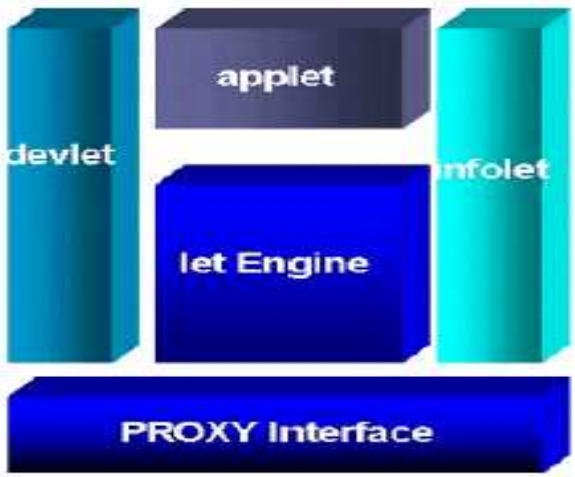


Figure 3: Proxy Platform

2.3 Proxy and Peer-to-Peer Networks

Peer-to-Peer technology is one of the most important technologies for ubiquitous computing since it supports one-to-one communication, free and extensible distribution of resources and an extensive support for the distributed search is already available for these networks through the distributed directory services. Proxy would play a major role in performing a happy marriage of mobile and Peer-to-Peer computing and thus providing platform for mobile Peer-to-Peer applications. Kato et al proposed a platform for mobile Peer-to-Peer communication [13] which treats the resource poor mobile clients as peers in Peer-to-Peer networks. A simple protocol for Peer-to-Peer communications with a mobile proxy is designed.

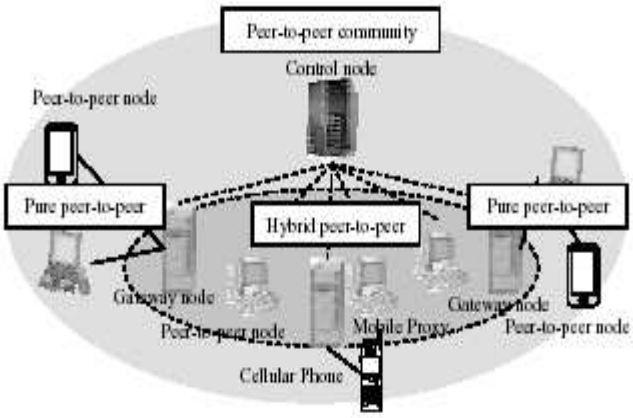


Figure 4: Proxy and Peer-to-Peer Networks

2.4 Proxy and Location Based Services

Location based Services need access to more location-sensitive information than the stationary computers. This challenge is to allow access to the information with out violating the users' privacy. Gerald proposes a proxy-architecture for location based services which gives the mobile device an ability to hide its location information or the personal identity of the mobile user [6]. Location based service proxy server acts as a SOAP dispatcher and a new server architecture called PELBPS (Privacy Enhanced Location Based Proxy Server) is presented. [5] Legitimate uses of location information could include contacting colleagues, routing telephone calls, logging meetings in personal diaries etc.

-

2.5 Proxy and Grid Computing

Proxy based middleware services for Peer-to-Peer computing in virtually clustered wireless grid networks were proposed by Junseok Hwang et al [12]. It is envisioned that the combination of mobile Peer-to-Peer applications and grid technologies could ultimately lead towards the goal of super computers with mobile devices anywhere, anytime. Figure 6 shows proxy-based wireless grid

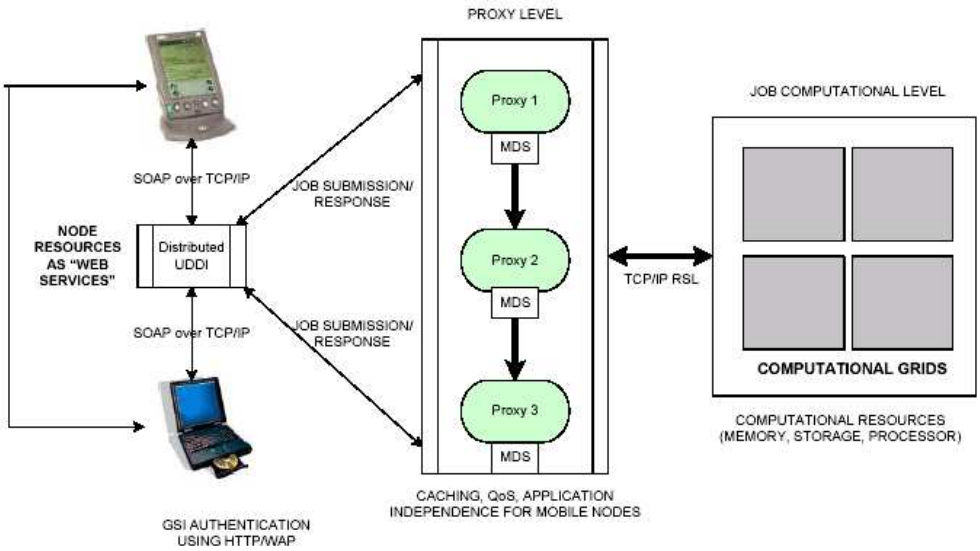


Figure 5: Proxy and grid computing

2.6 Proxy and Multimedia Applications

Proxies play an important role for streaming multimedia content over the internet. Multimedia proxy caching has not been sufficiently explored by the research communities. Infolibria MediaMall, RealSystem are examples for commercial multimedia proxy caches.

2.7 Proxy and Middleware

Use of proxy objects is prevalent in remote object interaction protocols. As an example, when an object needs to interact with a remote object, say across a network, the most preferred way of encapsulating and hiding the interaction mechanism is by using a proxy object that mediates communication between the requesting object and the remote object. To be specific, the stub and skeleton objects generated by the Remote Method Invocation (Java's Middleware) RMI compiler (rmic) are just local proxies for the real objects on the remote machines on client and the server respectively. In fact, these stubs and skeletons perform remote procedure call to send the request to the real object and then send the results back to the client on the local machine. Figure 8 refers to the proxy design pattern.

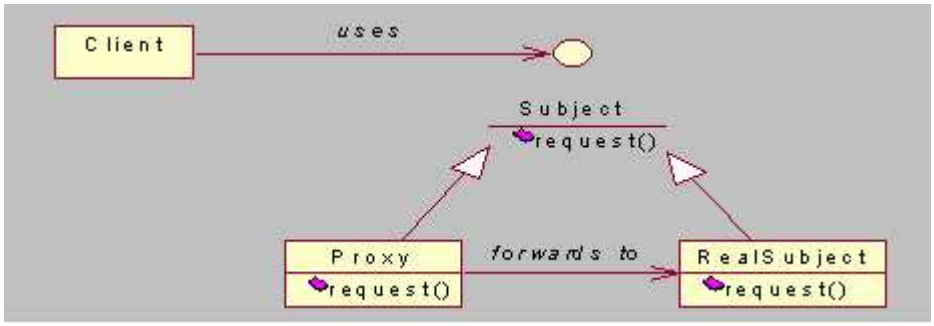


Figure 6: Proxy Design Pattern

2.8 Proxy as a Service

The following section describes the services that a proxy can offer.

1. A proxy can serve as a prefetching agent which prefetches data when the bandwidth is available. The prefetching technique can increase the hit rate considerably.

2. A proxy filter can result in more efficient use of networked resources, reduced costs and increased security.
3. Singh et al proposed a proxy that transcodes and caches. The excessive load of transcoding is pushed from server to proxy [22].
4. A proxy can be used to run an optimized protocol between itself and the mobile host [24]. An example could be Low Bandwidth X protocol, a compressed version of X11.
5. A proxy can support user and device profiles. This mapping at the proxy can support personalization of content, look-and-feel based on the users' device and interest.
6. The link characteristics could demand the proxies to be able to compress the data and send it to the mobile host. A proxy could be made more intelligent to drop, delay or send the data to the mobile host based on available bandwidth. Zenal et al discusses these functions of a proxy [24].
7. Gupta et al proposed ad insertion at proxies [8]. Some content providers disable caching at clients, since their business depends on the number of visits per day, the clients experience request latency due to increased traffic. If ads are inserted at proxies, the content providers can choose to insert different ads to different cached data streams.
8. Ads could be distracting to a user during his web experience. A client side proxy can be used to disable or remove the ads from web pages before they are presented to the user.

2.9 Advantages of using Proxies

File creation tends to be a frequent event and the high frequency of modification and accesses to modified files suggest that pre-fetching by proxies and push by servers proves to be useful [19]

Proxies assist in load balancing at servers and minimization of latency in delivery. They provide a good abstraction of the limitations of mobile devices. They result in systems which are

more fault-tolerant, scalable and robust. The functionalities of proxies described in section 3 show the advantages of using a proxy in a server.

2.10 Disadvantages of using Proxies

Proxy caching can be a solution to WWW traffic and server bottleneck problems. However, using a proxy as a caching entity between the client and server might have the following drawbacks.

1. The client might be looking at the stale data if the proxy is not updated [23].
2. A cache miss at the proxy might prove to be costly because the access latency might increase due to an intermediate entity between client and server [23].
3. Caching at proxies might reduce the number of hits on the original server which might lead to disabling caches by the web sites. This could result increased network traffic and congestion [4]. The hit rate of proxy caches is found to be low often not much higher than 50% [7]. A substantial percentage of cache miss is from the compulsory misses (First time accesses). Once a steady state has been achieved the hit rate would be greater.

2.11 Various Mathematical Distributions and their Implications for evaluating proxies

The mobile telephone call requests and WWW traffic were initially assumed to follow the Poisson or Markovian arrival process. However, the studies challenged that assumption and showed that the distributions often follow heavy tailed distributions (simplest being the Pareto distribution). A heavy-tailed distribution implies that regardless of the behavior of the distribution for small values of random variable, the asymptotic shape of the distribution is hyperbolic [3].

File popularity follows the Zipf's distribution where the popularity of the ith most popular file is proportional to (1/i). For instance, just the top 2% of documents could account for 90% of the accesses [19]

The file popularity distribution the documents accessed by mobile users do not follow the Zipf-like distribution [1]. However, the reason for deviation from Zipf-like distribution could because of a relatively small data set used for their analysis. It reported that users spend very less time on channels. This could be because of browsing on cell phones and PDAs could be cumbersome

and browse time on cellular networks is not free. Hence, in course of time with availability of better content, cheaper air time we could expect users to stay on the channels for longer time. It is reported that the WAP traffic shows strong resemblances to the self-similar property of the WWW traffic [15].

-

3. EXISTING ARCHITECTURES FOR MOBILE COMPUTING

In this chapter we present the existing architectures in wired and wireless worlds. We are presenting with five different architectures namely, thin-client, full client, client-proxy-server, hybrid proxy server and mobile proxy architectures followed by our proposed architecture called FTCPS which is flexible thick client proxy system.

3.1 Thin-Client Architecture (Proxy as an agent for Mobile Unit)

Figure 11 shows the block diagram of a thin-client architecture system in which a proxy is a computational agent for mobile device. Every request from the mobile device is handled by the mobile proxy. The request re-direction and processing is done at the server. The server becomes the bottleneck as all the requests pass through the server. This is a single point of failure in the system. If the server is down for some reason, entire system fails as a whole.

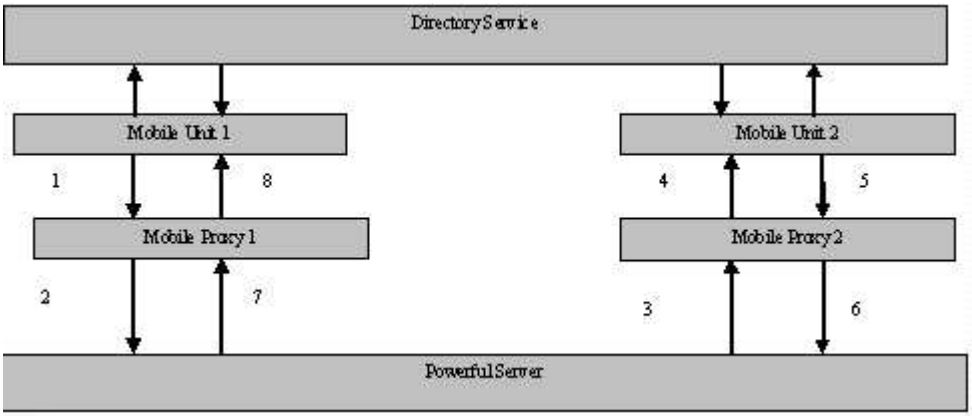


Figure 7: Thin-Client Architecture

Typical request-response flow in this architecture is as shown below.

1: Request from MobileUnit-1 to MobileProxy-1

2: Request from MobileProxy-1 to Directory Service

3. Request from Directory Service to MobileProxy-2

4. Request from MobileProxy-2 to MobileUnit-2

5. Response from MobileUnit-2 to MobileProxy-2

6. Response from MobileProxy-2 to Directory Service

7. Response from Directory Service to MobileProxy-1

8. Response from MobileProxy-1 to MobileUnit-1

Figure 12 shows the message sequence for the flow of events in a thin client system.

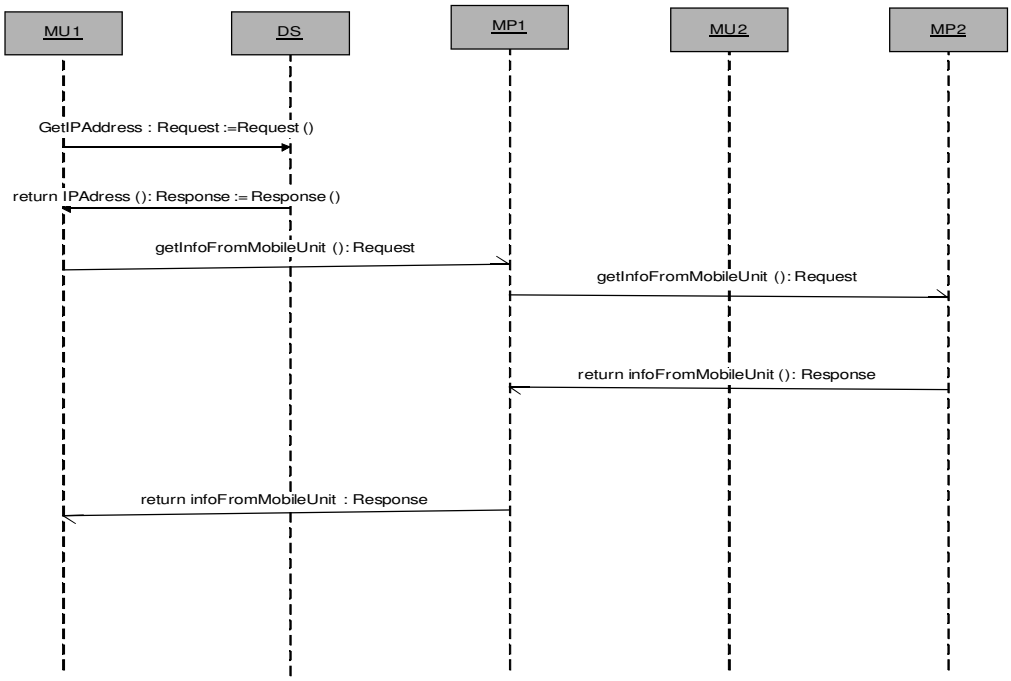


Figure 8: Sequence Diagram of Thin-Client Architecture

3.2 Full Client Architecture (No way to handle disconnections)

Figure 12 shows the block diagram of a full-client architecture system in which the client is self-sufficient in its computational power. It does not need a proxy to handle processing on its behalf. This is an ideal architecture for wired world where the disconnection probability is negligible. This architecture is handicapped by its inability to handle disconnections.

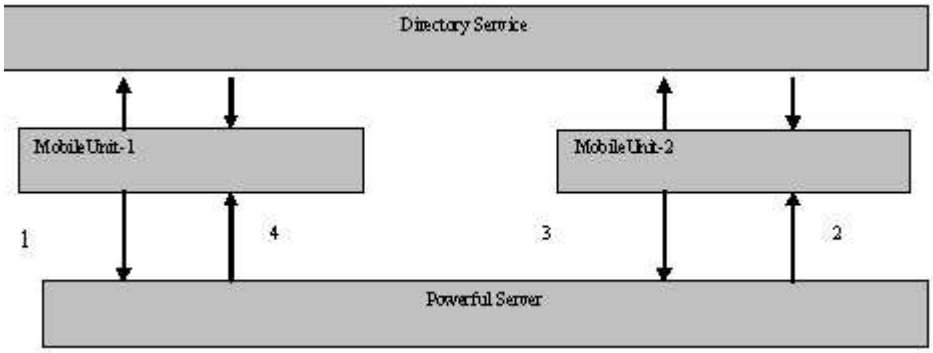


Figure 9: Full Client Architecture

Typical request-response flow in full-client architecture is as shown below.

1. Request from MobileUnit-1 to Server

2. Request from Server to MobileUnit-2

3. Response from MobileUnit-2 to Server

4. Response from Server to MobileUnit-1

Figure 14 shows the message sequence in a full client system.

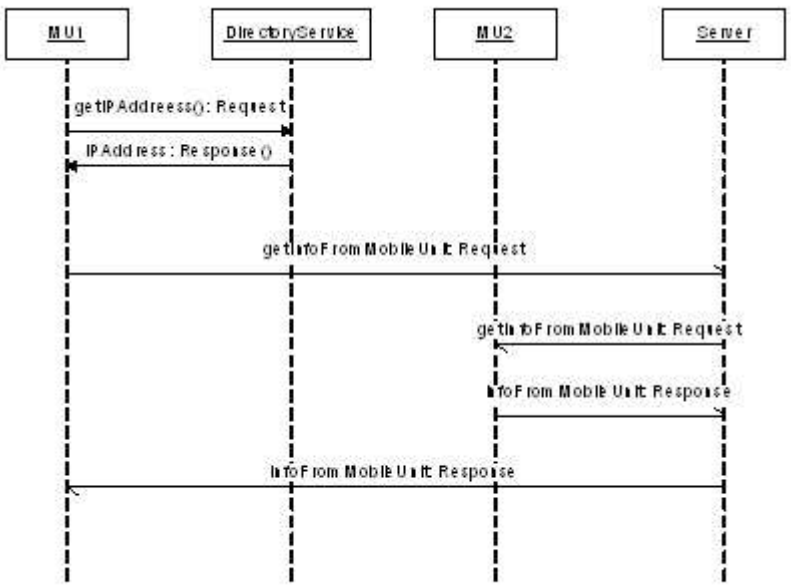


Figure 10: Message Sequence in Full Client Architecture

3.3 Client-Server with Proxy

Figure15 is a block diagram for the client-server with a proxy system. The proxy in this system handles disconnections. Apparent problem with this architecture is that the bandwidth is wasted if the mobile unit cannot handle the content given by the server when the mobile unit is not disconnected. If the server is given the responsibility to transcode the content according to the mobile unit's profile, the system becomes less scalable with server becoming the bottleneck.

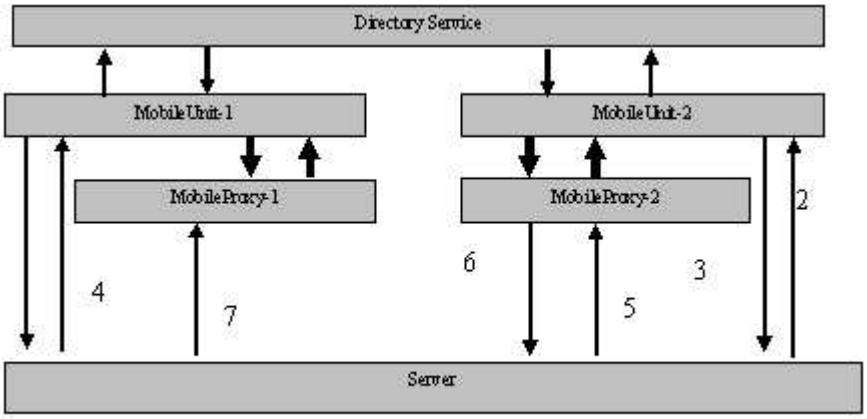


Figure 11: Client-Server with Proxy

Typical request-response flow in a client-proxy-server is shown below.

1. Request from MobileUnit-1 to Server
2. Request from Server to MobileUnit-2
3. Response from MobileUnit-2 to Server
4. Response from Server to MobileUnit-1
5. Request from Server to MobileProxy-2 (In case of disconnection of MobileUnit-2)
6. Response from MobileProxy-2 to Server
7. Response from Server to MobileProxy-1(In case of disconnection of MobiliUnit-1)

Figure 16 depicts the message sequence in a client-proxy-server system when the mobile unit and the proxy are synchronized and when the mobile unit cannot handle the content delivered by the server.

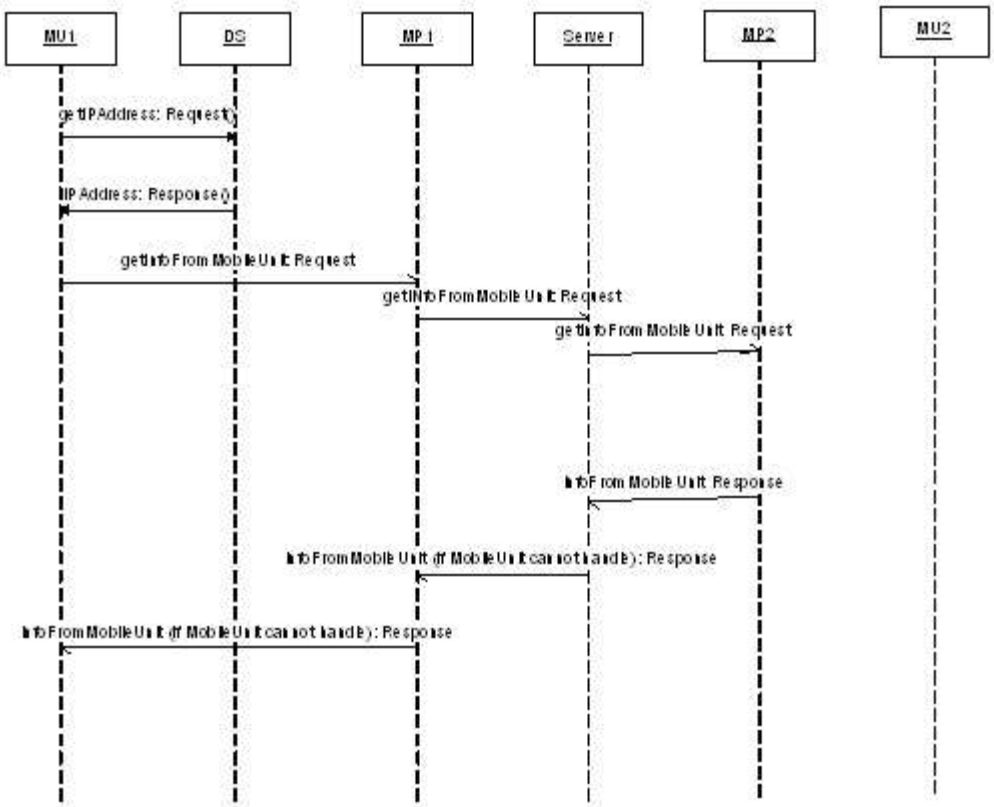


Figure 12: Message Sequence in Client Proxy Server System

Figure 17 depicts the message sequence in a client-proxy-server system when the mobile unit and the proxy are synchronized and when the mobile unit can handle the content delivered by the server.

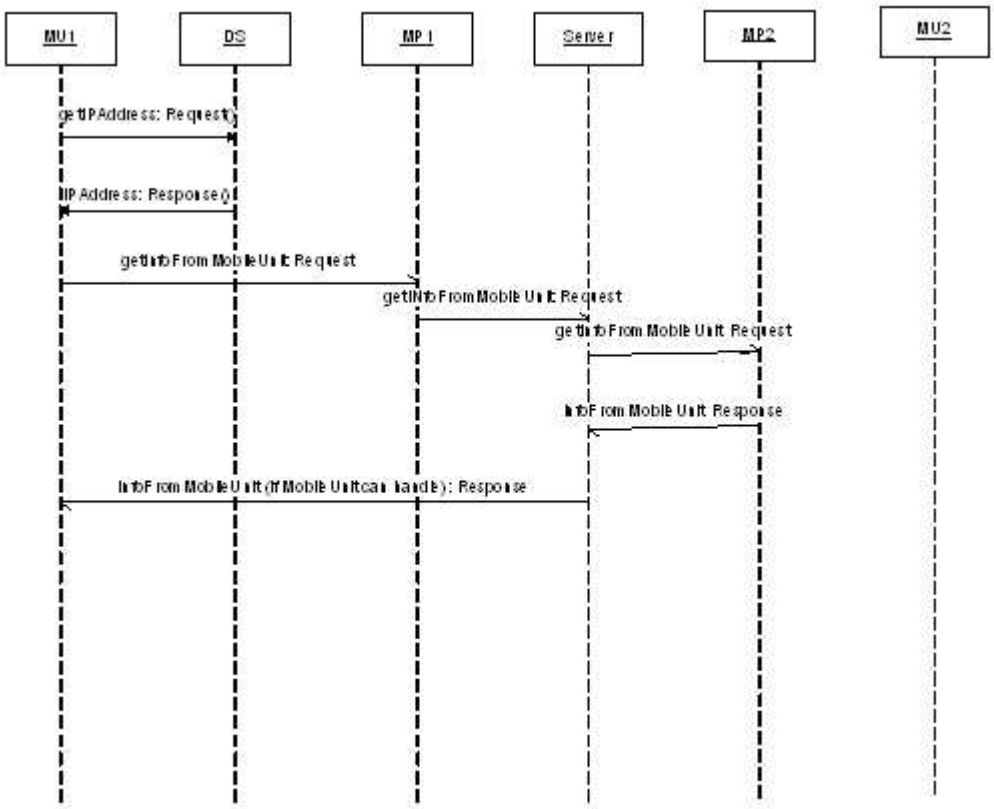


Figure 13: Message Sequence in Client Proxy System

3.4 Hybrid Client-Server with Proxy

More functionality at proxies makes the system less scalable [11]. Hence the server also shares the functionality of the proxy. Instead of pure proxy-based solution or pure end-end solutions, a hybrid approach is proposed which would scale better than the above two. Figure 18 shows the Hybrid-Client-Server with Proxy architecture proposed by Joshi et al [11].

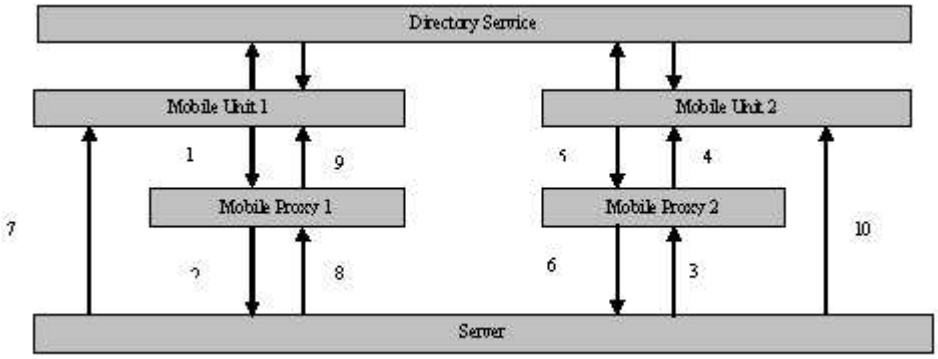


Figure 14: Hybrid Client-Server with Proxy

Typical request – response flow in the Hybrid Client-Server with Proxy architecture is as follows.

1. Request from MobileUnit-1 to MobileProxy-1
2. Request from MobileProxy-1 to Server
3. Request from Server to MobileProxy-2
4. Request from MobileProxy-2 to MobileUnit-2
5. Response from MobileUnit-2 to MobileProxy-2
6. Response from MobileProxy-2 to Server
7. Response from Server to MobileUnit-1 (if MobileUnit-1 can handle the response on its own)
8. Response from Server to MobileProxy-1 (if the MobileUnit-1 cannot handle the response on its own)
9. Response from MobileProxy-1 to MobileUnit-1
10. Request from Server to MobileUnit-2 (if MobileUnit-2 can handle the request)

-

Figure 19 depicts the message sequence for a synchronized mobile unit and proxy and when mobile unit cannot handle the content delivered by the server

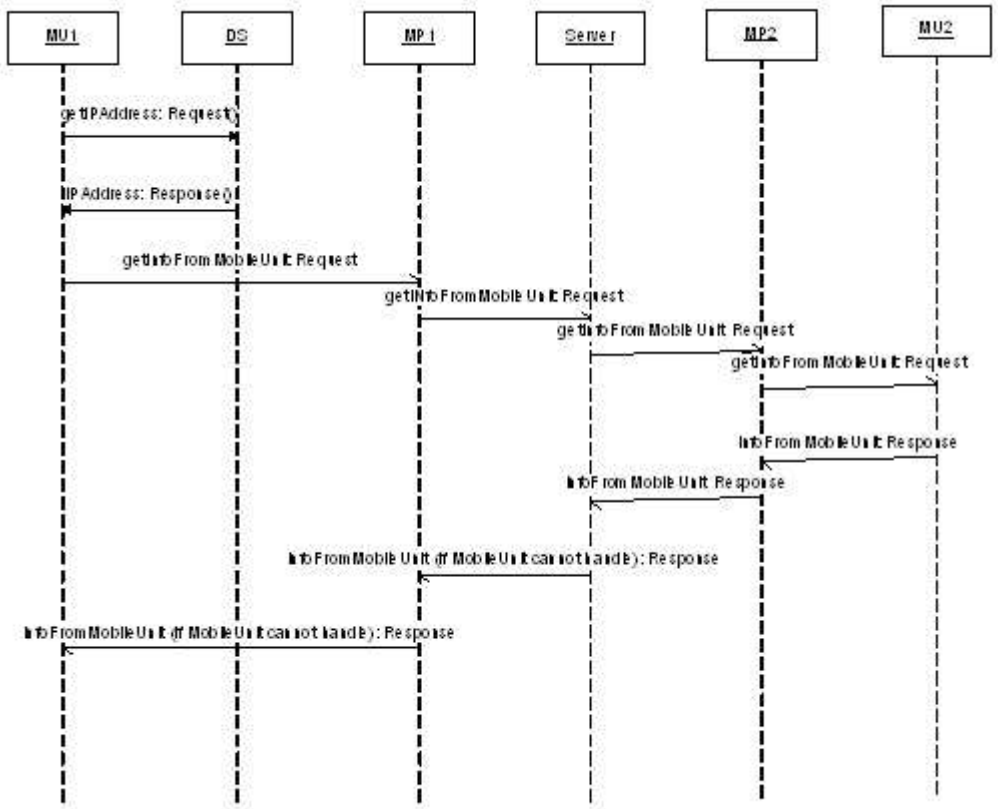


Figure 15: Message Sequence in Hybrid Client Proxy Server System

Figure 20 depicts the message sequence diagram for a synchronized mobile unit and proxy that are not synchronized and when mobile unit can handle the content delivered by the server

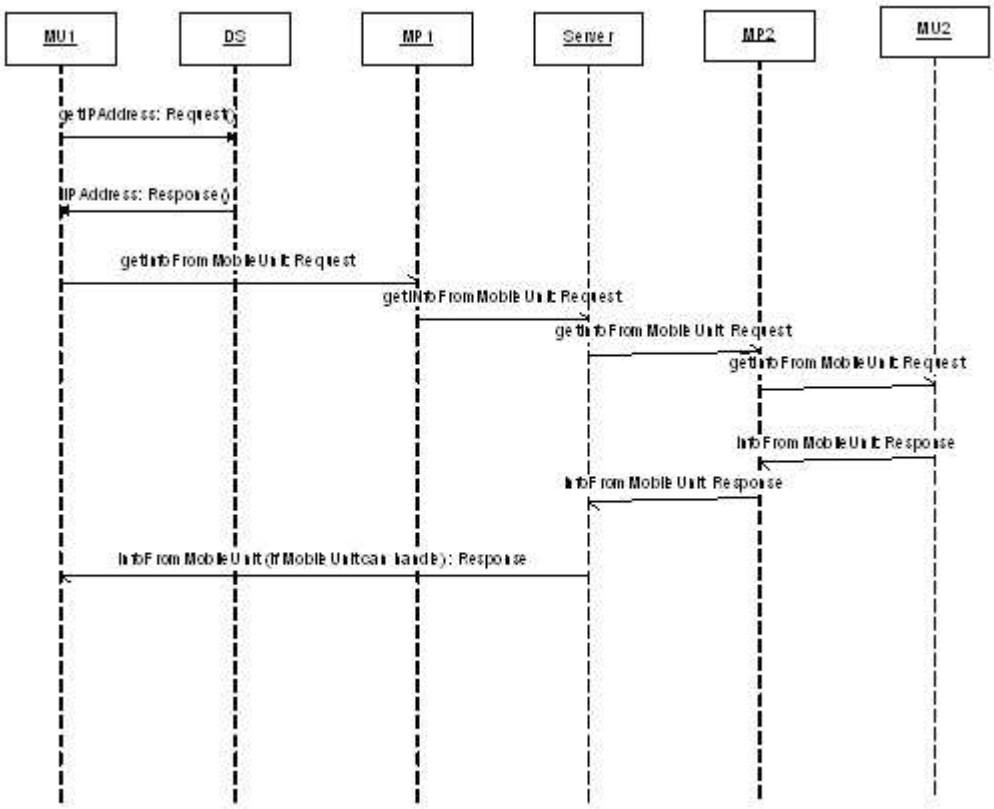


Figure 16: Message Sequence in Hybrid Client Proxy Server System

3.5 Flexible Thick Client Proxy System (Our Proposal)

The synergy between networking and mobility will engender new collaborative applications with mobile devices on heterogeneous platforms. One such middleware is "SYSTEM ON MOBILE DEVICES", SYD developed by the Yamacraw Embedded Systems research team. SYD is a new paradigm for collaborative application development. SyD consists of a combination of middleware and clientware technologies that enables application development independent of the specific nature of the device, data, services, or network (device locations) that the application is targeted to run over [26]. SyD enables such independent application development through well defined wrapping standards for devices, data, and services.

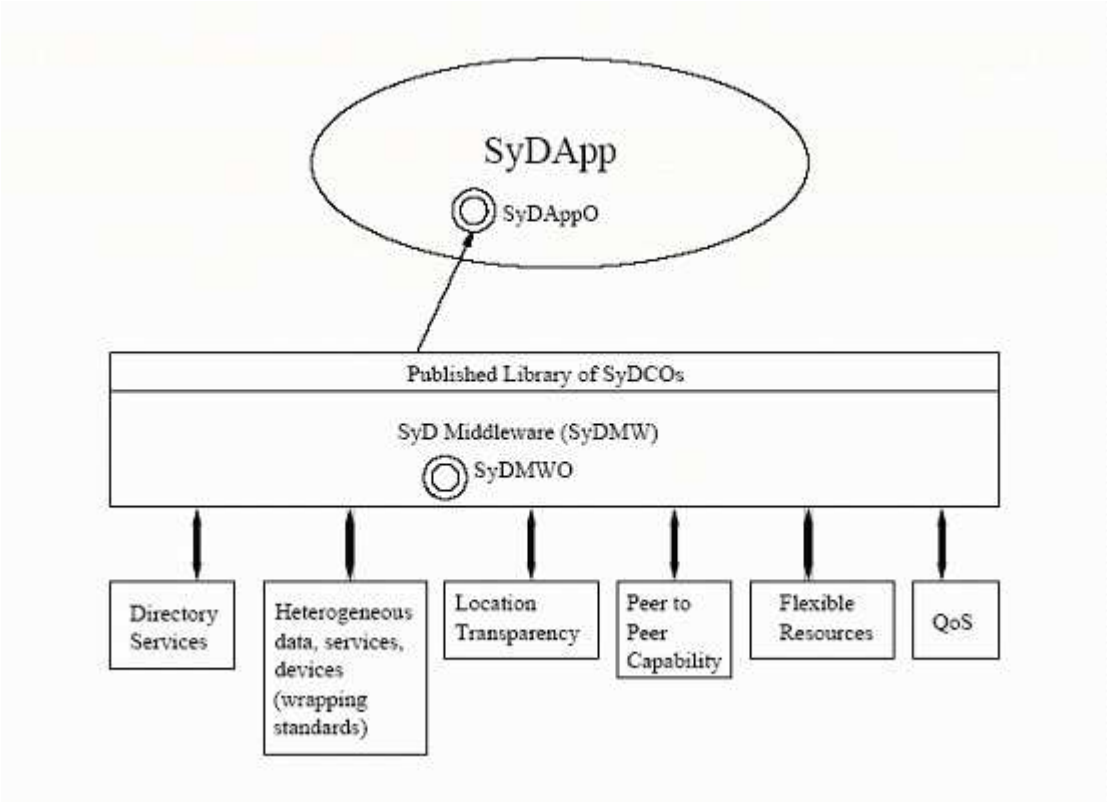


Figure 17: SyD Architecture

Following questions need to be answered while designing architecture for mobile applications using a proxy.

Should a proxy be monolithic or distributed in design?

What level of transparency to disconnections is to be guaranteed?

How to increase the scalability of the system?

How to make the system more robust and fault tolerant?

What functionalities could be offloaded to clients from proxies?

FTCPS is the proposed proxy architecture that supports that the peer to peer capability for mobile networks. The following block diagram explains the FTCPS – Flexible Thick Client Proxy System. Mobile Units are peers and can communicate with each other. If a mobile unit is disconnected during the communication, then its proxy plays the role of a mobile unit for serving the request. For example, if Mobile Unit – 1 is making a request to Mobile Unit -2 which is disconnected then, Mobile Proxy 2 serves Mobile Unit-1's request and if Mobile Unit

-

-1 is disconnected while Mobile Proxy -2 is serving, then the response is sent to Mobile Proxy -1.

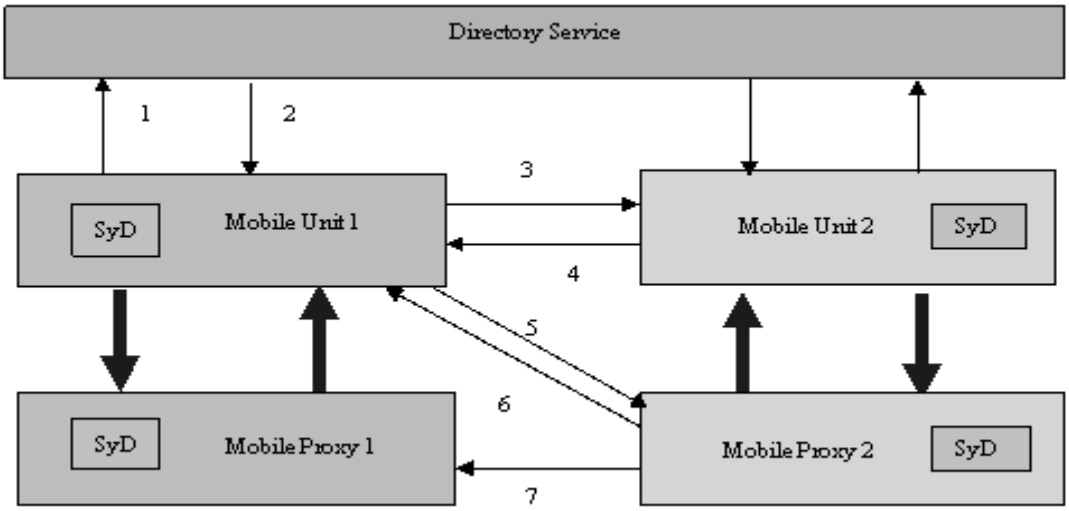


Figure 18: Flexible Thick Client Proxy System

Typical request-response sequence in FTCPS is as follows.

1: Request from MobileUnit-1 to Directory Service

2: Response from Directory Service to Mobile Unit-1

3. Request from MobileUnit-1 to MobileUnit-2

4. Response from MobileUnit-2 to MobileUnit-1

5. Request from MobileUnit-1 to Mobile Proxy-2 (If MU2 is disconnected)

6. Response from Mobile Proxy-2 to MobileUnit-1

7. Response from MobileProxy-2 to Mobile Proxy –1 (If MobileUnit-1 is down, Response is redirected to MobileProxy-1)

-

Figure 23 shows the message sequence in FTCPS when MU1 and MU2 are both connected. The sequence of events is as follows.

1: Request from MobileUnit-1 to Directory Service

2: Response from Directory Service to Mobile Unit-1

3. Request from MobileUnit-1 to MobileUnit-2

4. Response from MobileUnit-2 to MobileUnit-1

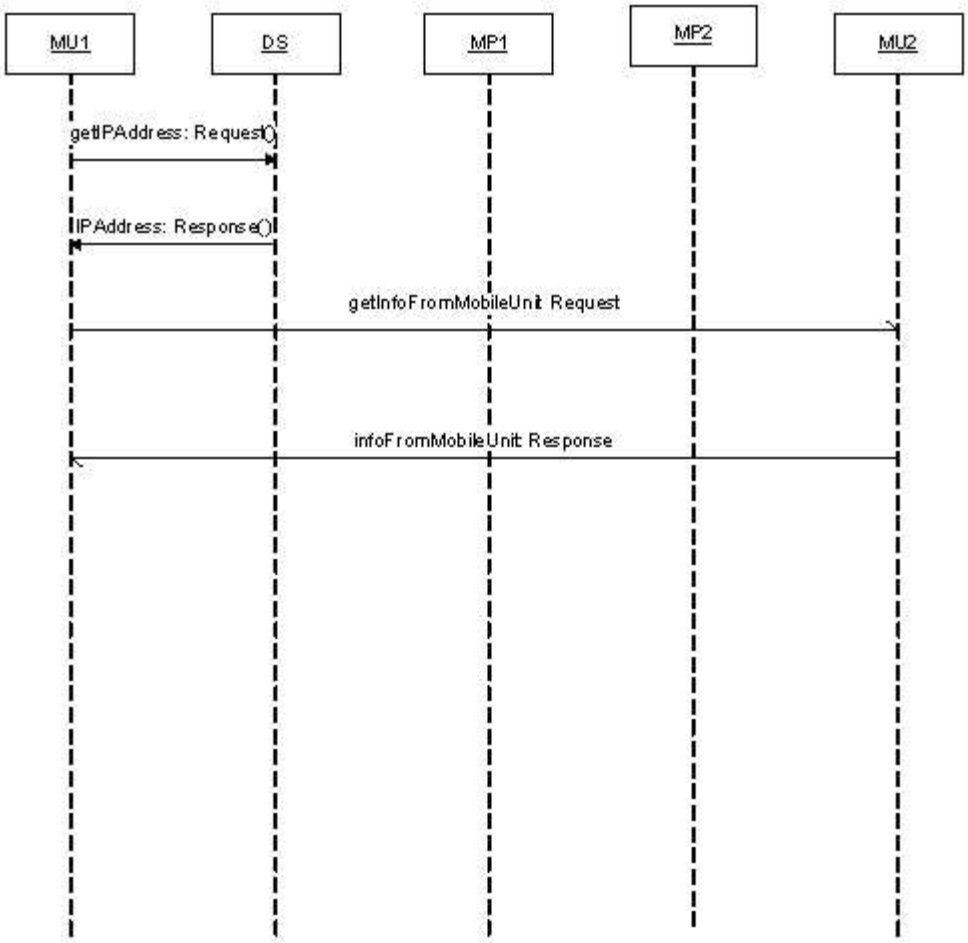


Figure 19: Message Sequence in FTCPS when MU1 and MU2 are both connected

-

Figure 24 shows the message sequence in FTCPS when MU2 is disconnected. The request-response sequence is as follows.

1: Request from MobileUnit-1 to Directory Service

2: Response from Directory Service to Mobile Unit-1

3. Request from MobileUnit-1 to MobileProxy-2

4. Response from Mobile Proxy-2 to MobileUnit-1

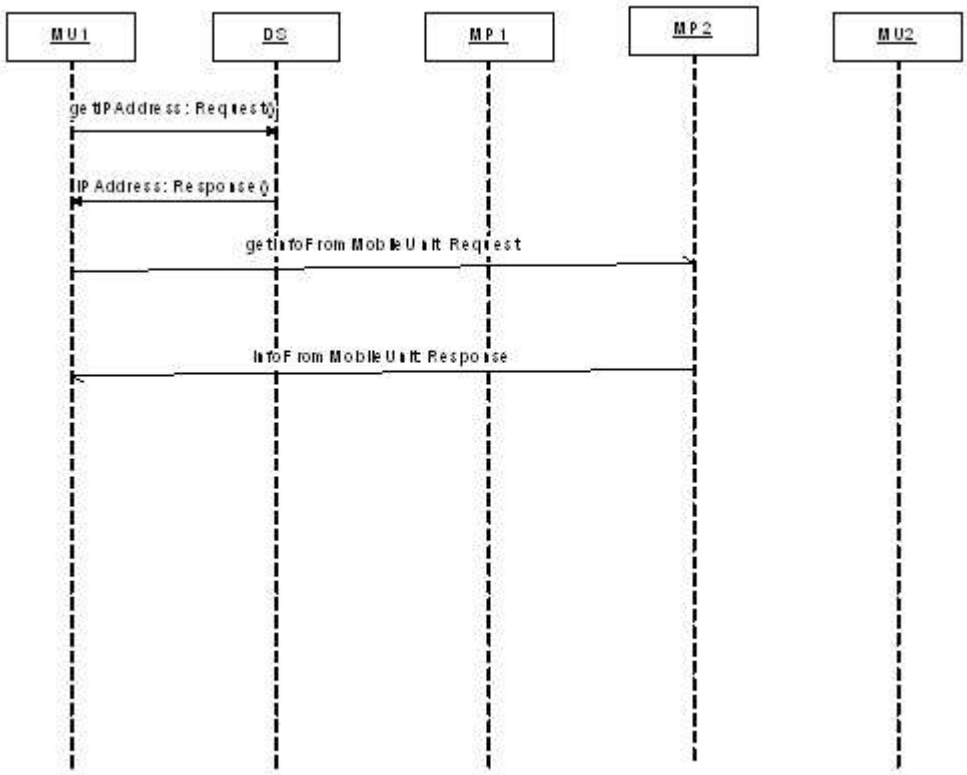


Figure 20: Message Sequence in FTCPS when MU2 is disconnected.

Figure 25 shows the message sequence in FTCPS when MU1 is disconnected after making the request to MU2. The request-response sequence is as follows.

1: Request from MobileUnit-1 to Directory Service

2: Response from Directory Service to Mobile Unit-1

3. Request from MobileUnit-1 to MobileUnit-2

4. Response from MobileUnit-2 to MobileProxy-1

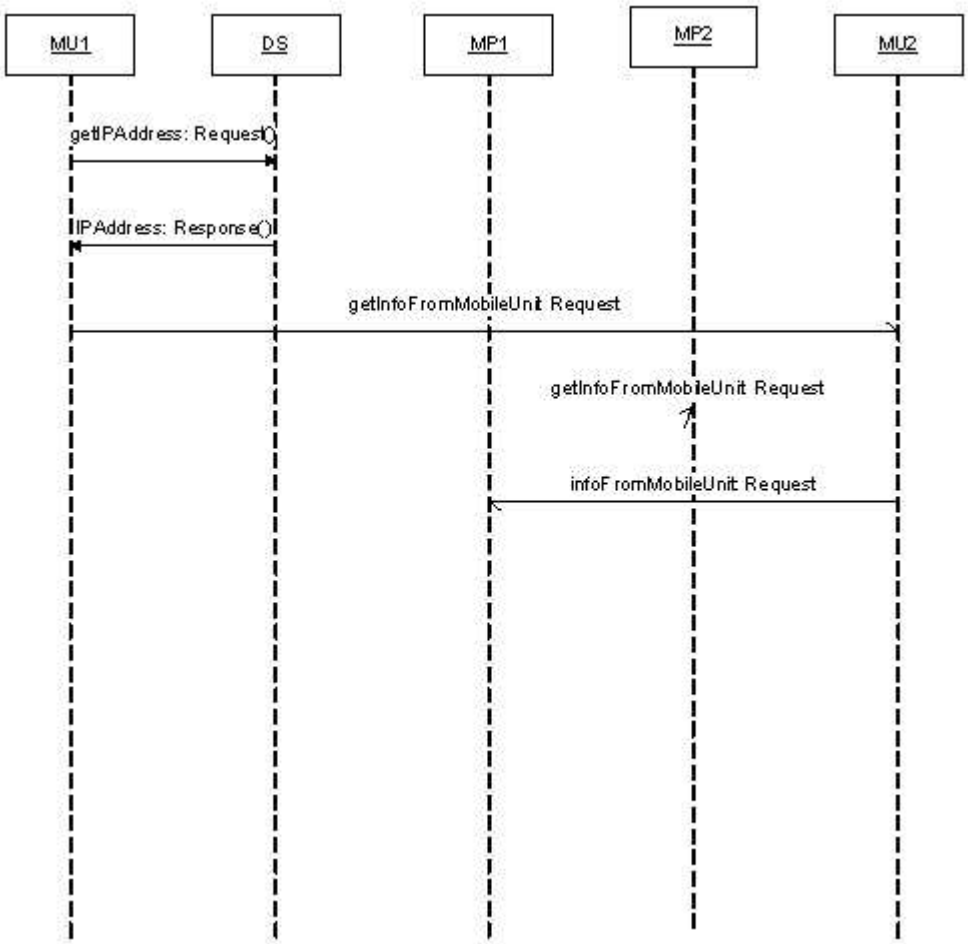


Figure 21: Message Sequence in FTCPS when MU1 is disconnected after making the request to MU2

Figure 26 shows the message sequence in FTCPS when MU1 is disconnected after making the request to MU2 which is disconnected. The request-response sequence is as follows.

1. Request from MobileUnit-1 to Directory Service

2. Response from Directory Service to Mobile Unit-1

3. Request from MobileUnit-1 to Mobile Proxy-2

4. Response from MobileProxy-2 to Mobile Proxy -1

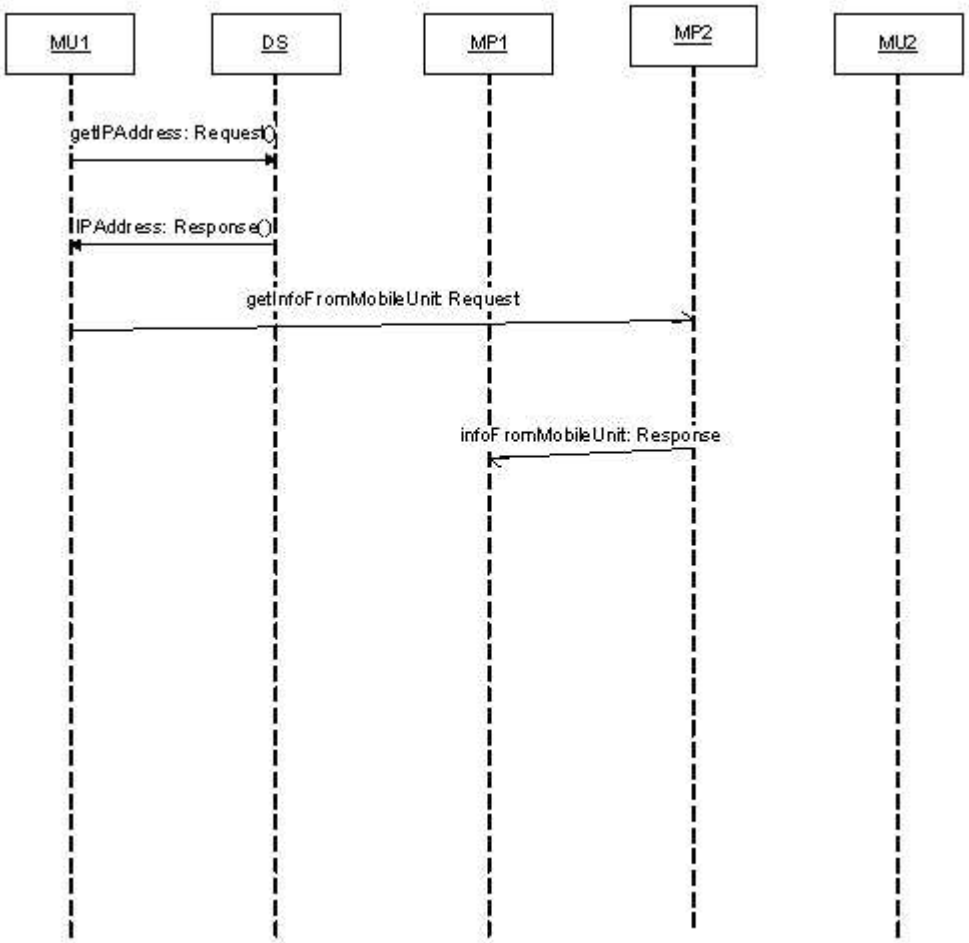


Figure 22: Message Sequence when MU1 is disconnected after making the request to MU2 which is disconnected

A slight variation of FTCPS called Hybrid-FTCPS has an intelligent proxy which can hide the mobile unit's incapability to handle some responses. The following block diagram explains the Hybrid-FTCPS – Flexible Thick Client Proxy System. Mobile Units are peers and can

communicate with each other. If a mobile unit cannot handle the content delivered by the responding mobile unit or proxy, then the proxy of the mobile unit plays the role delivering the content that could be handled by the mobile-unit. This means that in FTCPS, a proxy can come handy when a mobile proxy is disconnected or when a mobile unit cannot handle the content delivered by the responding proxy or the server. A typical example could a user requesting a PDF file but do not have acrobat reader to view the document. The proxy could be more intelligent here to either deliver the content in a format that the mobile unit can accept or drop the request as it is a waste of bandwidth to deliver the content that cannot be handled by the mobile unit.

4. SIMJAVA: DESIGN CHOICE FOR MODELING THE SIMULATION

SIMJAVA became our natural choice for simulation as it has the following features to bolster our experimental set-up.

1) Support for JAVA
2) In built support for Statistical Analysis
3) Package of Mathematical Distributions
4) Graphs generating package
5) Efficient TRACE tools
6) Animation package

It is very costly to measure the scaling-up and scaling-down factors in a system in terms of infrastructure. So simulations of large scale systems are preferred and are studied by incorporating real-time parameters from the study of existing real time systems. Our experiments needed events which followed various mathematical distributions, effective tracing tools, and a tool which eases the analysis of the results obtained. SIMJAVA was strongly preferred as it had all the features that this experimental set-up demanded.

4.1 What is SIMJAVA?

SIMJAVA is a toolkit for building working models of complex systems. It is based around a discrete event simulation kernel and includes facilities for representing simulation objects as animated icons on screen. SIMJAVA simulations may be incorporated as ``live diagrams" into web documents. SIMJAVA is actually a collection of three packages, `eduni.simjava`, `eduni.simanim` and `eduni.simdiag`. `eduni.simjava` is a package for building stand alone text only java simulations, which produces a trace file as the output by default. `eduni.simanim` is tightly integrated with the text only simulation package, and provides a skeleton applet for easily building a visualization of a simulation. `eduni.simdiag` is a collection of JavaBeans based classes for displaying simulation results.

Using a programming language to build models (rather than building them graphically) has the advantage that complex regular interconnections are straightforward to specify, which was

crucial for some of the networks we were interested in simulating. It also allows the inclusion of existing libraries of code to build simulations.

The SIMJAVA package has been designed for simulating fairly static networks of active entities which communicate by sending passive event objects via ports. This model is appropriate for hardware and distributed software systems modeling.

4.2 The SIMJAVA Design

A `SIMJAVA` simulation contains a number of entities each of which runs in parallel in its own thread. An entity's behavior is encoded in Java using its `body ()` method. Entities have access to a small number of simulation primitives:

- `sim_schedule ()` sends event objects to other entities via ports.
- `sim_hold ()` holds for some simulation time.
- `sim_wait ()` waits for an event object to arrive.
- `sim_select ()` selects events from the deferred queue.
- `sim_trace ()` writes a timestamped message to the trace file.

In `SIMJAVA` event objects are passed to other entities via ports using `sim_schedule ()`. They are automatically queued, and retrieved as required by the receiver using `sim_select ()` and `sim_wait ()`. `sim_select ()` is used to select from events which have already arrived, and `sim_wait ()` waits for the next future event.

The other difference from message passing interaction models is that in `SIMJAVA` all events are globally sorted by simulation timestamp to ensure that messages never arrive out of order.

4.3 Overview of SIMJAVA

This section gives an introduction to SIMJAVA and a detailed API is available at http://www.icsa.informatics.ed.ac.uk/research/groups/hase/simjava/

This is an excerpt taken from the SIMJAVA tutorial[25]

4.3.1 Setting up the simulation

The simulation is managed by `Sim_system`, a static class which will be setup in our `main ()` method. To define the simulation's `main ()` method we will have to create one further class. The name given to this class should be representative of the system being simulated and also be given to the file containing all the classes.

In any simulation the following four steps are required:

1. Initialize `Sim_system`.

2. Make an instance for each entity.

3. Link the entities' ports.

4. Run the simulation.

4.3.3 What is the trace of a simulation?

Simulations are often quite complex programs. Whenever complexity is added to any program the number of errors present is bound to increase. After building a simulation it is always good practice to test it before fully instrumenting it with statistical measures and exhaustively running it. A tool that is very useful in a simulation's debugging process (also known as verification) is the simulation's trace. The trace is essentially internal information that is made available to the modeler through which the exact actions within the simulation can be examined. Such an examination could lead to the discovery of undesired entity behavior which would require the simulation's modification.

-

4.3.4 Sampling distributions

It is more suitable to describe an entity's performance characteristics by means of certain distributions; sampling them to produce specific values for e.g. a disk's seek time. By using distributions the real world is more accurately simulated since the entities' behavior will be, as in the real world, stochastic. A good example of the importance of non-determinism is a network router whose packet inter-arrival times are far from deterministic.

In order to generate samples from a distribution a random number generator needs to be used. This generator is sampled to produce a sample uniformly distributed between `0` and `1`. Following this, the uniform sample is modified to fit in the desired distribution. This process is called random variate generation.

To be precise, the random number generators used in simulation packages are pseudorandom. This means that although a generator's output is statistically random, if it is setup in the same way at a later time it will produce the exact same sequence of samples. This is of great importance in simulation studies since without pseudorandomness experiments would not be repeatable. Furthermore, maintaining the exact same sequence of samples enables the modeler to focus on the effect of changes introduced to the system. If for example a modification to the simulation is made and different results are observed, this difference will be totally attributed to the modifications and not to the random samples used.

4.3.5 Adding statistical measurements

The first step to adding statistical support to entities is to define measures of interest. In order to understand how measures are defined we need to identify their possible types. These types will dictate how they are defined, updated, and used to produce measurements.

All measures can be classified into three categories:

- Rate based measures.
- State based measures.
    - Continuous.
    - Non-continuous.
- Interval based measures.

Rate based measures are based on the occurrence of an event over a period of time. When the sums of these events are taken into account along with the interval in which they occurred. Such measures are for example an entity's throughput or loss rate. In the first case the event of interest is an event completing all service and in the second case, an event having been lost (according to a user defined condition). State based measures reflect the entity's state over a period of time. An entity can be considered as being in one state for a certain interval and then being in another for another interval. A state based measure can be considered as continuous if the entity moves from one state to another in a continuous fashion. An example of this could be the entity's utilization in which case the entity is either busy or not, moving seamlessly between these two states. A non-continuous state based measure does not have the characteristic of continuity. This means that each interval is not required to begin where the previous one ended but may begin at a later time. Interval based measures usually have less to do with the entity itself and more to do with events that pass through it. Such measures reflect time intervals that were experienced by these events. Examples of such measures could be the waiting time or residence time of events at an entity.

-

4.3.6 What is a transient condition?

Simulations are started off in an arbitrary starting state. In this starting state entities quite often exhibit a different behavior compared to the behavior they would have if the simulation progressed for a while. Once the bias of the system's original state is overcome, the system is considered to have entered steady state. In this state the entities' behavior remains largely the same.

It is often the case that simulations are built to study the system only after it has warmed up i.e. reached steady state. This, for example is the approach used by Markovian modeling techniques that solve global balance equations to obtain the steady state probability distribution. The period of time from the beginning of the simulation up to the point that steady state starts is termed the transient or warm-up period. The effects of this period should be discarded if steady state analysis is of interest. Alternatively, if the modeller is interested in a system's startup, the transient period should be included in calculations.

In simulations the transient period can't be mathematically identified. As such, the modeller needs to specify a condition after which steady state is considered to have been entered. This condition is termed here the transient condition

4.3.6 Defining a transient condition

A transient condition is set using a `set_transient_condition()` method when the simulation is being setup. Three types of transient conditions are available:

- Event completions: The system is considered to have entered steady state after a number of event service completions at a given entity. To use this method the modeller needs to specify the entity of interest, an event tag to identify events counting towards the condition, and a number of event completions.

- Elapsed time: In this case the transient period is explicitly identified as a time period from the beginning of the simulation. The modeller needs to provide a point in time after which the system is considered to have entered steady state.

- The minimum-maximum method: This method allows `Sim_system` to attempt to automatically locate the time at which the transient period has elapsed. To do this the modeller provides an entity and one of its measures. The minimum-maximum method searches the given measure's observations and decides that the transient period has elapsed once an observation is located that is neither the minimum nor the maximum of the remaining observations. This method is quite crude and in most cases fails to identify a long enough transient period.

- None: in this last case, any transient condition is specified. This is used when transient analysis is of interest and is the default when the `set_transient_condition ()` method isn't used.

4.3.7 Defining a termination condition

A termination condition is quite obviously the condition that once satisfied, terminates the simulation. As in the case of transient conditions, since version 2.0, SIMJAVA centrally holds and checks the termination condition. A termination condition is defined in a similar way to the transient one by using a `set_termination_condition ()` method. The following termination condition types are available to the modeller:

-

- Event completions: Similar to the corresponding transient condition type. The modeller provides an entity and an event type tag, as well as the number of event completions. Once the number of specified events have completed the simulation will terminate.

- Elapsed time: The termination time is explicitly set. The modeller provides a point in time that once reached, will terminate the simulation.

- Confidence interval accuracy: A very useful termination condition. This termination condition tells `Sim_system` to run until a measure's total mean has been calculated to a certain degree of accuracy. To use this method the modeller provides an entity and one of its measures, a confidence level, an accuracy level, and an output analysis method to use for variance reduction. The total mean differs from the sample mean and is the mean produced after applying output analysis to the simulation. The accuracy level is the ratio of the confidence interval half width of the specified measure's mean, over the total mean. The confidence interval will be calculated with the confidence level provided. More on this termination condition type will be discussed in the relevant section covering output analysis.

- None: As in the case of the transient condition, this it the default if a `set_termination_condition ()` method isn't used. Running an infinite simulation may seem pointless but may be of some use if animation is used. In this case, the simulation's animation could be used as an elaborate demo of a system.

5. EXPERIMENTAL SET UP FOR SIMULATION AND RESULTS

We carried out our experiments in the following environment.

JAVA: Programming language for the simulation.

SIMJAVA: Package for simulation.

PERL: To parse the SIMJAVA trace file and put into an excel sheet

UNIX: For large scale simulations of 500 nodes

WINDOWS: For small scale simulations of 100-300 nodes

5.1 Thin Client Architecture

In a thin client system, a mobile unit has computation limitations. Each mobile unit has a proxy which is its computation agent and a gateway for flow of request and responses. For example, a cell phone which has low memory and low battery life has a proxy at the base station. When a cell phone is out of charge, the proxy would store messages for the device at the base station. The simulation has four entities namely, Mobile Unit, Mobile Proxy (Proxy for the mobile unit), Directory Service and a Server. All these are subclasses of Sim_entity. The simulation requires establishing the links among the entities before starting the experiment. This means that the network designed for the experiment is static in terms of number of entities involved; however, the probability of disconnection makes the network dynamic. A random number generator is used to decide the destination entity of a request. Since we have a centralized server, connecting any two devices through this server's ports is easy. In this set-up, since a mobile proxy is the computation agent for the device, the responses are served by the proxy. A mobile device can make requests of following four types

1) File handling requests
2) Database requests
3) Method invocation requests
4) HTTP requests

-

The program also takes into account the probability of disconnections. The probability of each type of requests can be varied. The experiments are carried out for networks of 100, 200, 300, 400 and 500 nodes.

Table 1 show the trace statistics for an experiment where the http and method invocation requests are predominant.

Table 1: Trace Statistics

Request Type	Number of Requests	Percentage
HTTP	20000	40
File Handling	5000	10
Method Invocation	15000	30
Database	5000	10
Total	50000	100

We evaluated the performance of mobile units, proxies and servers using a combination of simulation and experimental evaluation. In the simulation study, we modelled various architectures using SIMJAVA. We used the SIMJAVA's event generator to generate the events which follow POISSON and NEG-EXP distributions. These events are translated into file system operations, http requests, database operations and method invocation which are sent to other entities through the Sim_ports. In all the experiments, we feed the simulator with a trace of 50000 requests summarized by the table below with trace statistics. The performance metric recorded was the residence time. To distinguish between the events arrived at the server uniquely; we identified each event with the timestamp of its arrival. We used a Hashmap to store the source and destination of that event appended by its timestamp. An event arrived at server has the information about the source of the event, the timestamp and the data packet. We had placeholders for the information about source, destination and the current entity. In thin-client architecture, we do not need to vary the disconnection probability to study the system because the requests from client pass through proxy always. So the experiments were conducted on this system only varying the number of nodes as shown below.

-

Table 1 shows the residence time of proxy in thin client architecture. The residence time increases from 507 seconds to 800 seconds suggested by the slope of the graph in Figure 28 when the size of the network is increased from 100 to 500 nodes.

Table 1: Residence Time for Proxy in Thin Client Architecture

	100	200	300	400	500
Proxy	507	600	700	750	800

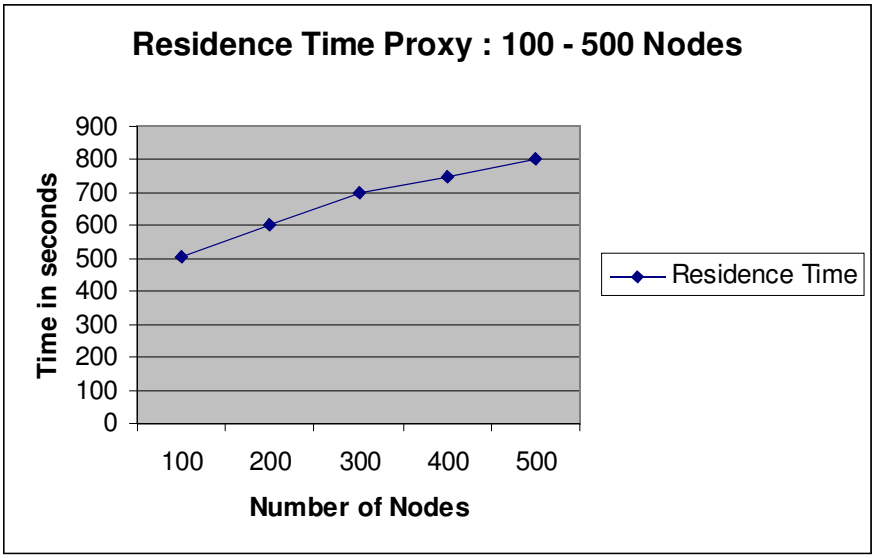


Figure 23: Residence Time of a Proxy in Thin Client Architecture

Table 2 shows the residence time of client in thin client architecture. The residence time increases from 485 seconds to 508 seconds suggested by the slope of the graph in Figure 29 when the size of the network is increased from 100 to 500 nodes.

Table 2: Residence Time of Client in Thin Client Architecture

	100	200	300	400	500
Client	485	495	500	505	508

-

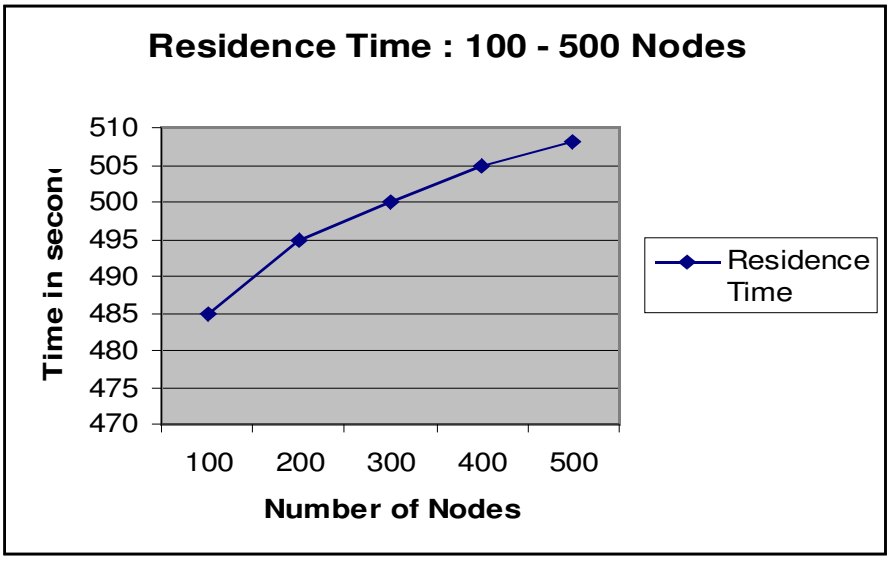


Figure 24: Residence Time of a Client in Thin Client Architecture

Table 3 shows the residence time of server in thin client architecture. The residence time increases from 464 seconds to 23809 seconds suggested by the steep slope of the graph in Figure 30 when the size of the network is increased from 100 to 500 nodes. This clearly shows that the server is the bottleneck that affects the scalability of this system.

Table 3: Residence Time of Server in Thin Client Architecture

	100	200	300	400	500
Server	464	9373	14170	18647	23809

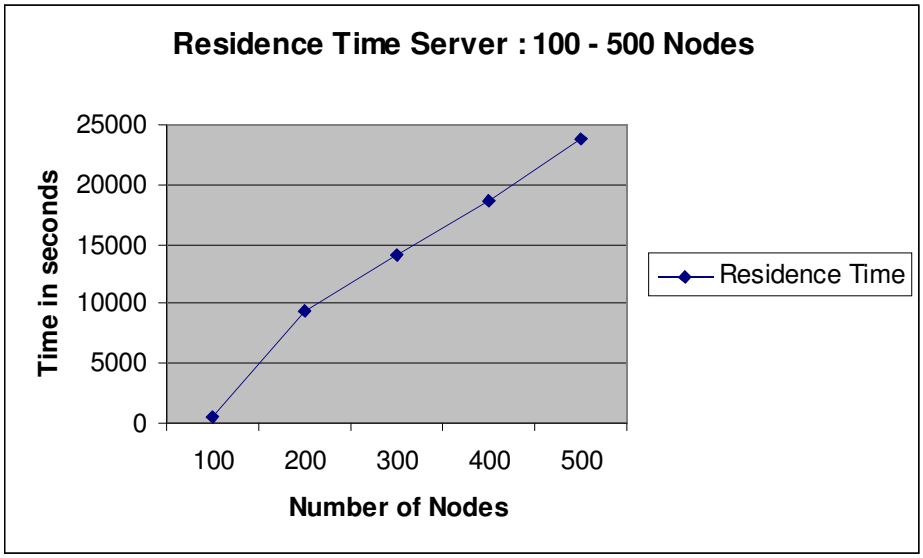


Figure 25: Residence Time of a Server in Thin Client Architecture

5.2 Full-Client Architecture

Full Client Architecture is one extreme in the spectrum of client-server computing. A full client does not need a proxy to be its computational agent. Typically, Directory service is a process at the server. We separated the directory service from the server to depict the steps in making a request clearly. The Full Clients are connected to a centralized server and the communication among the clients is through the server. The blatant defect of this model is its intolerance to disconnections. The system is not scalable. Hence this architecture is not suitable for mobile Peer-to-Peer networks. Our experiment set up has full clients connected to directory service and the server. The simulated system is an exact representation of the block diagram above. Since there is no way to handle disconnections, all the requests from FC_x to FC_y are lost if FC_y is disconnected and all the responses from FC_y to FC_x are lost if FC_x is disconnected. We designed a parent entity which as placeholder for source, destination and the name of the current entity. The system was studied for various probabilities of disconnections as summarized in Table 4. Figure 35 represents the graphical view of results in Table 4.

Table 4: Residence Time in Full Client Architecture

	30	40	50	70	80	100
Client	1014	1121	1084	1158	1189	1190
Server	671	1035	1619	629	338	338

-

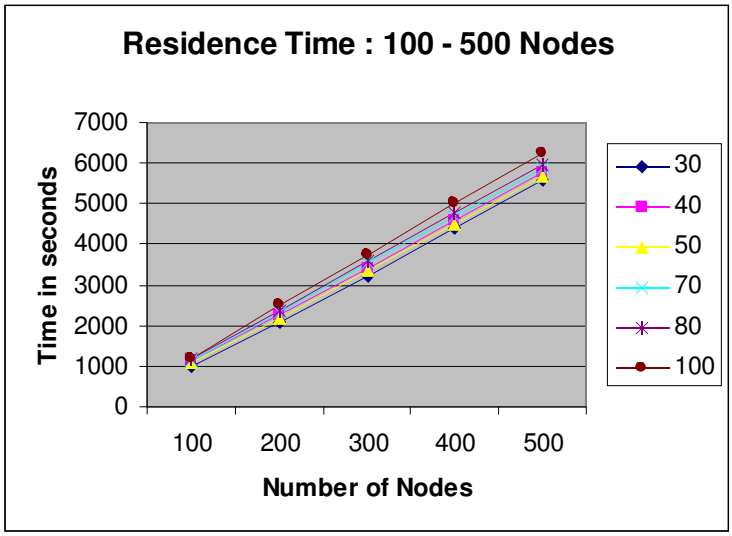


Figure 26: Residence Time of 100 - 500 Nodes

5.3 Client Proxy Server System

In a Client Proxy Server system, the disconnections of client are handled by proxy in the system. Table 5 shows the results of residence time of proxy in this system. Figure 36 is the graphical representation of the results. It is evident from Figure 36 that the residence time of proxy increases with the increase in the probability of disconnection.

Table 5: Residence Time of Proxy in Client Proxy Server System

Residence Time Proxy	30	40	50	70	80
100	182.00	312.00	415.00	577.00	720.00
200	409.00	601.00	800.00	1223.00	1364.00
300	615.00	910.00	1127.00	1775.00	2015.00
400	829.00	1202.00	1507.00	2356.00	2705.00
500	1132.00	1606.00	1986.00	2957.00	3452.00

-

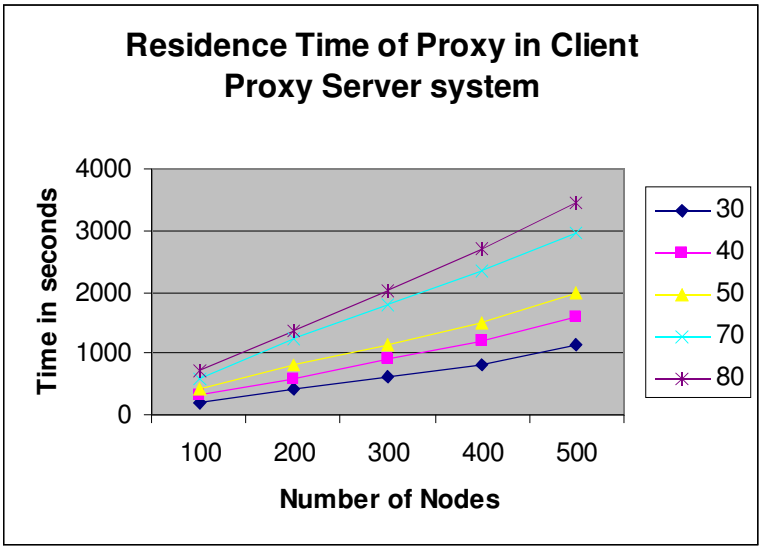


Figure 27: Residence Time of Proxy in a Client Proxy Server System

Table 6 shows the results of residence time of client in client proxy server system. Figure 37 is the graphical representation of the results. It is evident from Figure 36 that the residence time of client decreases with the increase in the probability of disconnection.

Table 6: Residence Time of Client in Client Proxy Server System

Residence Time Client	30	40	50	70	80
100	181.00	83.00	23.00	23.00	14.00
200	360.00	130.00	55.00	18.00	30.00
300	537.00	183.00	102.00	25.00	30.00
400	629.00	202.00	150.00	30.00	33.00
500	713.00	301.00	220.00	33.00	35.00

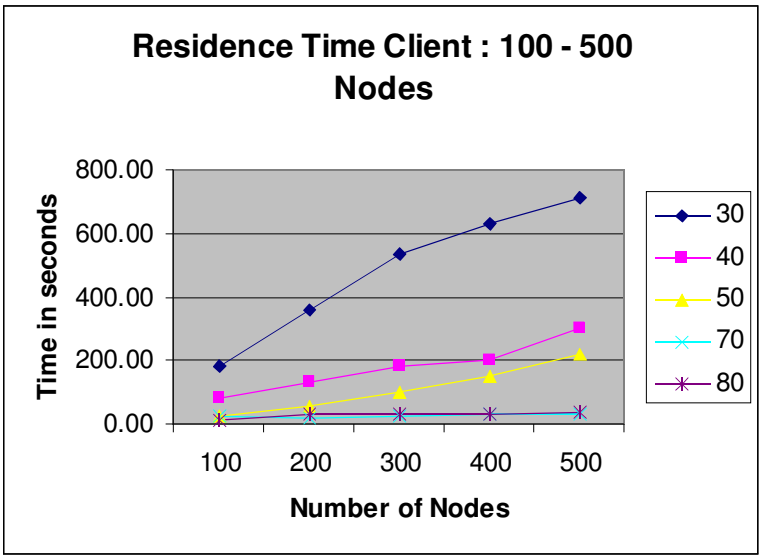


Figure 28: Residence Time of Client in Client Proxy Server System

Table 7 shows the results of residence time of server in client proxy server system. Figure 38 is the graphical representation of the results. It is evident from Figure 36 that the residence time of server increases with the increase in the probability of disconnection and with the increase in number of nodes.

Table 7: Residence Time of Server in Client Proxy Server System

Residence Time Server	30	40	50	70	80
100	166.00	243.00	270.00	154.00	294.00
200	327.00	485.00	577.00	586.00	608.00
300	509.00	736.00	852.00	905.00	921.00
400	609.00	948.00	1213.00	1168.00	1190.00
500	821.00	1242.00	1458.00	1517.00	1511.00

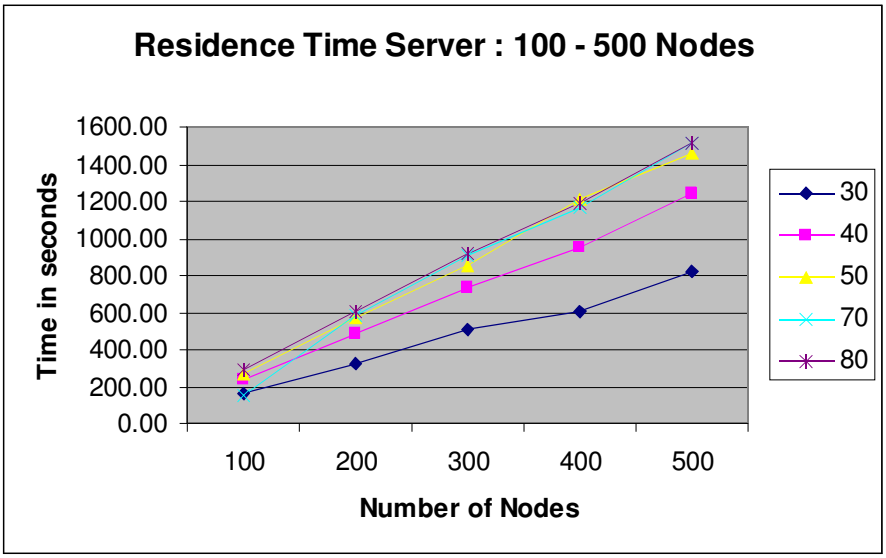


Figure 29: Residence Time Server 100 - 500 Nodes

5.4 Flexible Thick Client Proxy Server

To present a realistic model for a calendar application, we came up with 125 clusters of 4 nodes in each cluster. Since this is a scenario for mobile Peer-to-Peer networks, proxy handles the disconnections for mobile units. Hence each cluster would be a complete graph with 4 mobile units and 4 mobile proxies. This implies that there are 56 (8*7=56) bidirectional links to be established. In SIMJAVA, each bidirectional link is programmed as two unidirectional links.

-

A HashMap is an ideal data structure to keep track of the links in a graph. We used 4 HashMaps to capture the links of all the 4-node clusters in the network. Typical entity in a HashMap is shown in Table 8.

Table 8: Data Structure(HashMap) for the links in the network cluster

From Port	To Port
MU1	MU2
MP1	MP2
MP1	MU2
MU1	MP1

We used four Hashmaps to capture entire network.

1) HMUtoU : For the links between Mobile Unit to Mobile Unit
2) HMUtoP : For the links between Mobile Unit to Mobile Proxy
3) HMPtoU : For the links between Mobile Proxy to Mobile Unit
4) HMPtoP : For the links between Mobile Proxy to Mobile Proxy

Trace and sim reports are flat files generated for each experimental set-up. We needed a way to compare results by parsing these flat files. A PERL script is written to parse the sim_report file and export the results to an excel sheet. Table 9 shows the residence time of a client in FTCPS for disconnection probabilities ranging from 30 percent to 100 percent for a network of size from 100 nodes to 500 nodes. Figure 44 shows the graphical view of client's residence time in FTCPS when the network is scaled up from 100 nodes to 500 nodes for various disconnection probabilities.

Table 9: Residence Time of Client in FTCPS

	30	50	70	80	100
100	63	71	75	78	78
200	114	131	146	154	156
300	169	196	220	229	234
400	222	258	292	314	312
500	276	324	366	379	390

-

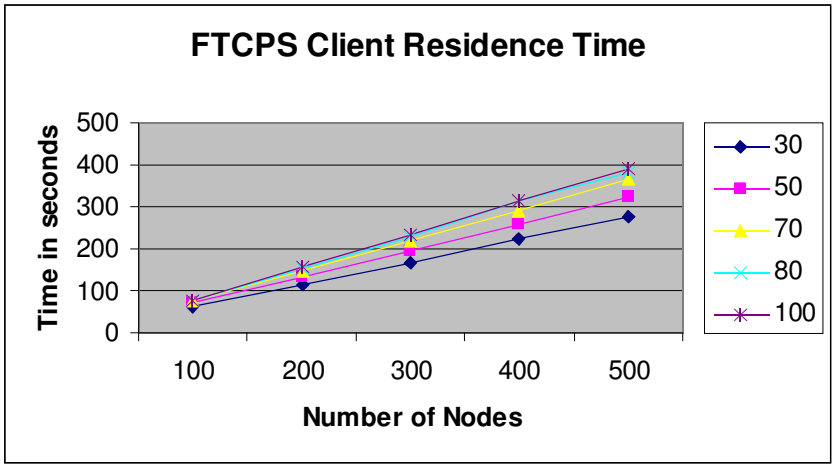


Figure 30: FTCPS Client Residence Time 100 – 500 Nodes

Table 9 shows the residence time of a proxy in FTCPS for disconnection probabilities ranging from 30 percent to 100 percent for a network of size from 100 nodes to 500 nodes. Figure 45 shows the graphical view of proxy's residence time in FTCPS when the network is scaled up from 100 nodes to 500 nodes for various disconnection probabilities. The residence time of proxy in FTCPS rises very slightly with increase in number of nodes.

Table 10: Residence Time of Proxy in FTCPS

	30	50	70	80	100
100	6	8	11	13	16
200	5	8	10	11	13
300	5	7	10	11	14
400	5	7	10	11	14
500	5	7	10	11	14

-

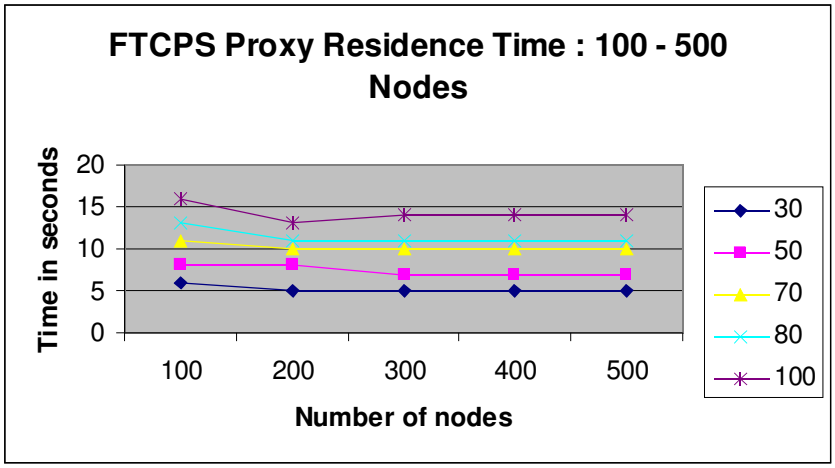


Figure 31: FTCPS Proxy Residence Time 100 - 500 Nodes

Table 11 shows the residence time of a server in FTCPS for disconnection probabilities ranging from 30 percent to 100 percent for a network of size from 100 nodes to 500 nodes. Figure 46 shows the graphical view of server's residence time in FTCPS when the network is scaled up from 100 nodes to 500 nodes for various disconnection probabilities. The residence time of server in FTCPS is constant even when the network is scaled up.

Table 11: Residence Time of Server in FTCPS

	30	40	50	70	80	100
100	1	1	1	1	1	1
200	1	1	1	1	1	1
300	1	1	1	1	1	1
400	1	1	1	1	1	1
500	1	1	1	1	1	1

-

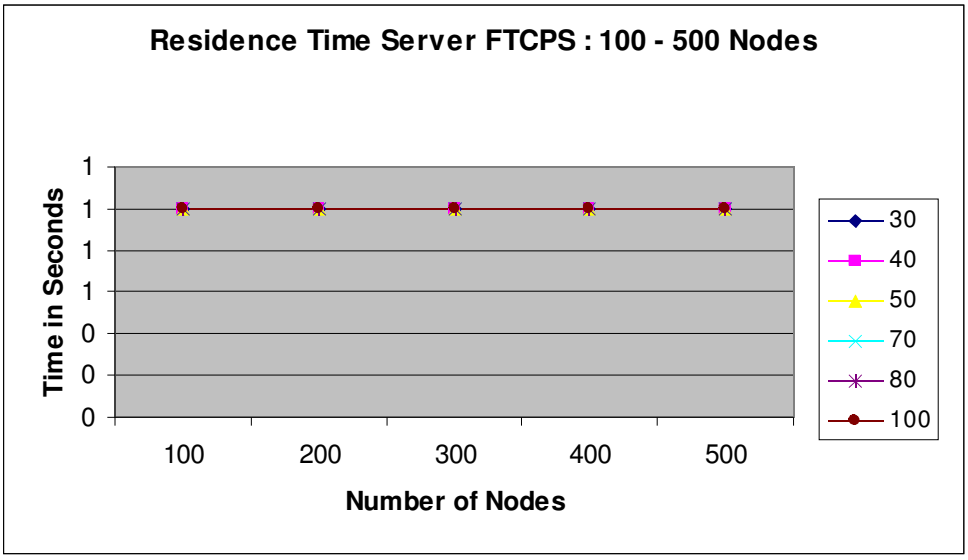
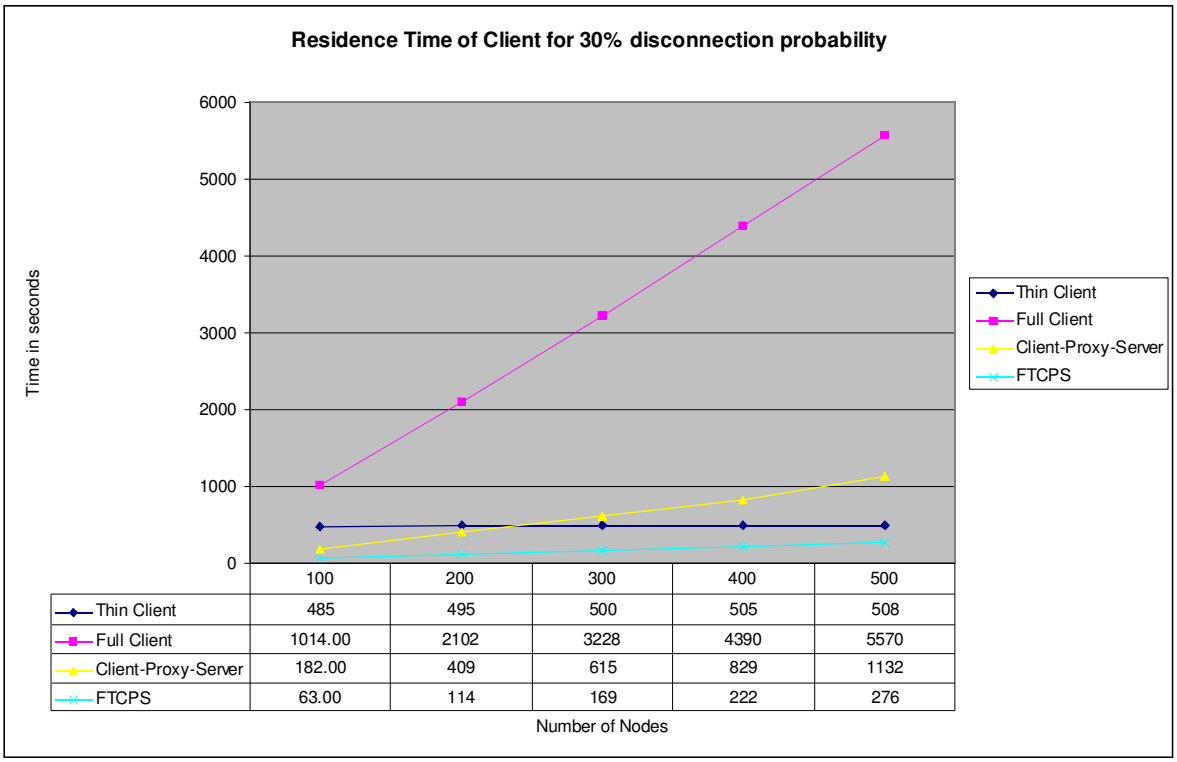


Figure 32: FTCPS Server Residence Time 100 – 500 Nodes

Figure 47 shows the residence time of events at clients for different architectures for 30% disconnection probability. It is evident from figure 47 that the residence time of client in FTCPS is the least and did not increase steeply with increase in the number of nodes. The residence time of client in Full Client architectures had the maximum residence time. The behavior of Client Proxy Server system and FTCPS is similar at lower disconnection rates.

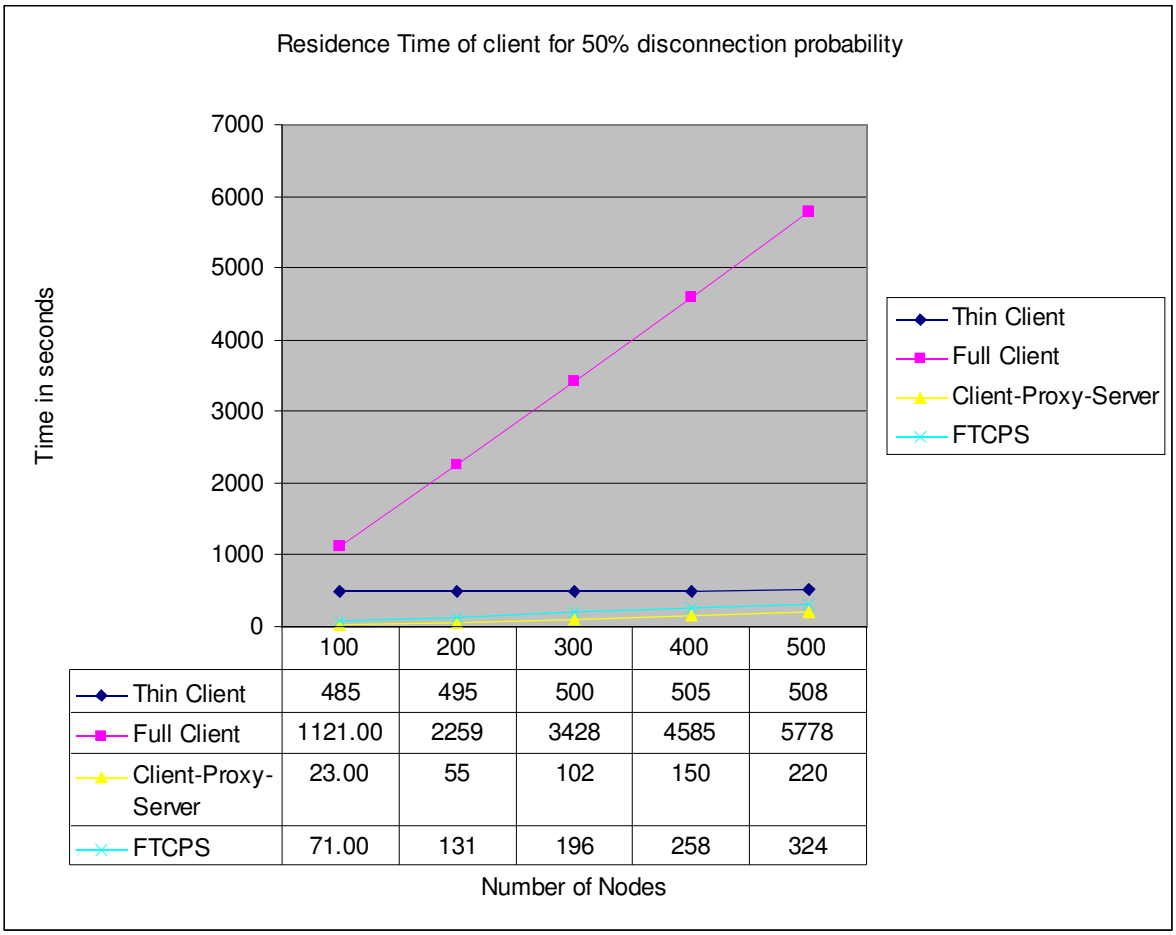


	100	200	300	400	500
Thin Client	485	495	500	505	508
Full Client	1014.00	2102	3228	4390	5570
Client-Proxy-Server	182.00	409	615	829	1132
FTCPS	63.00	114	169	222	276

Figure 33: Residence Time of Client for 30% disconnection probability

-

Figure 48 shows the residence time of events at clients for different architectures at 50% disconnection probability. The residence time of client in Full Client architectures had the maximum residence time. The resident time of Client Proxy Server system and FTCPS is similar at 30% disconnection probability.

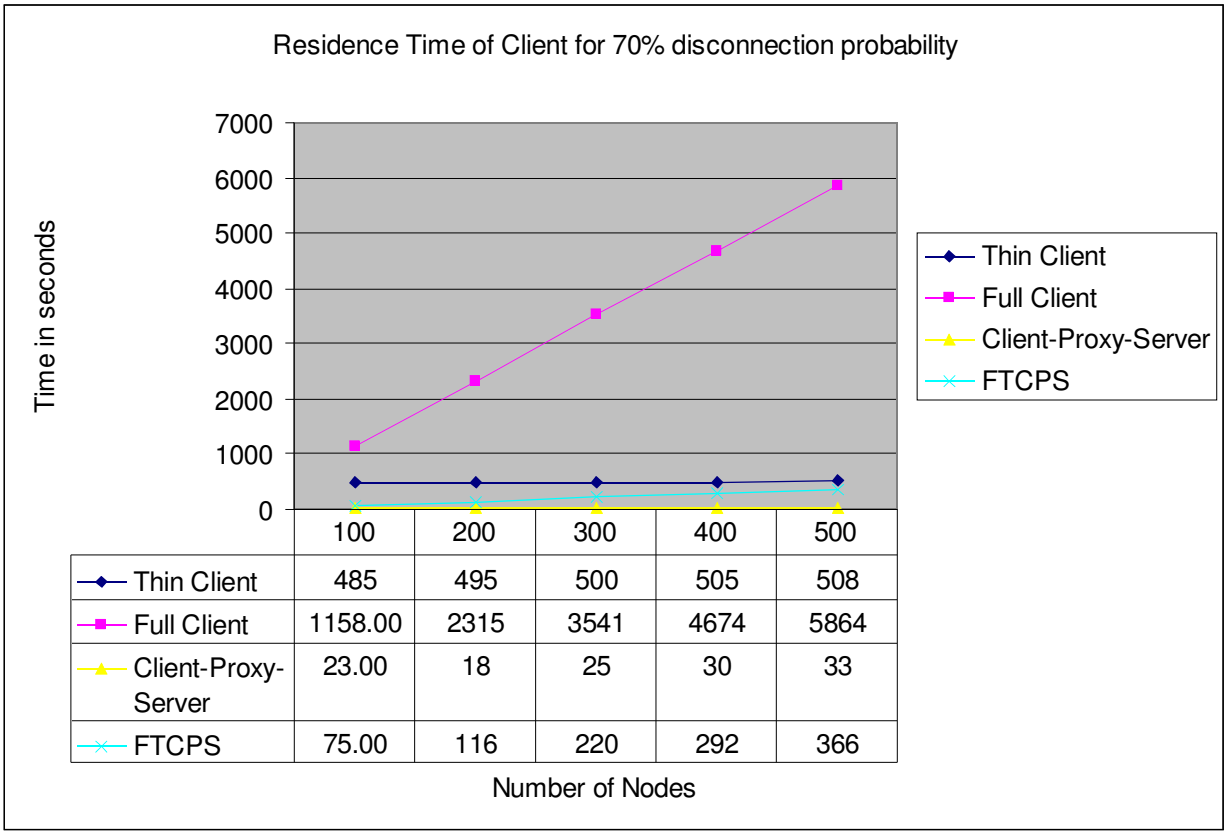


	100	200	300	400	500
Thin Client	485	495	500	505	508
Full Client	1121.00	2259	3428	4585	5778
Client-Proxy-Server	23.00	55	102	150	220
FTCPS	71.00	131	196	258	324

Figure 34: Residence Time of Client for 50% disconnection probability

-

Figure 49 shows the residence time of events at clients for different architectures at 70% disconnection probability. The client in Full Client architecture has the maximum resident time and the Client Proxy Server systems' client has the least resident time since the events end up at proxy if client is disconnected.

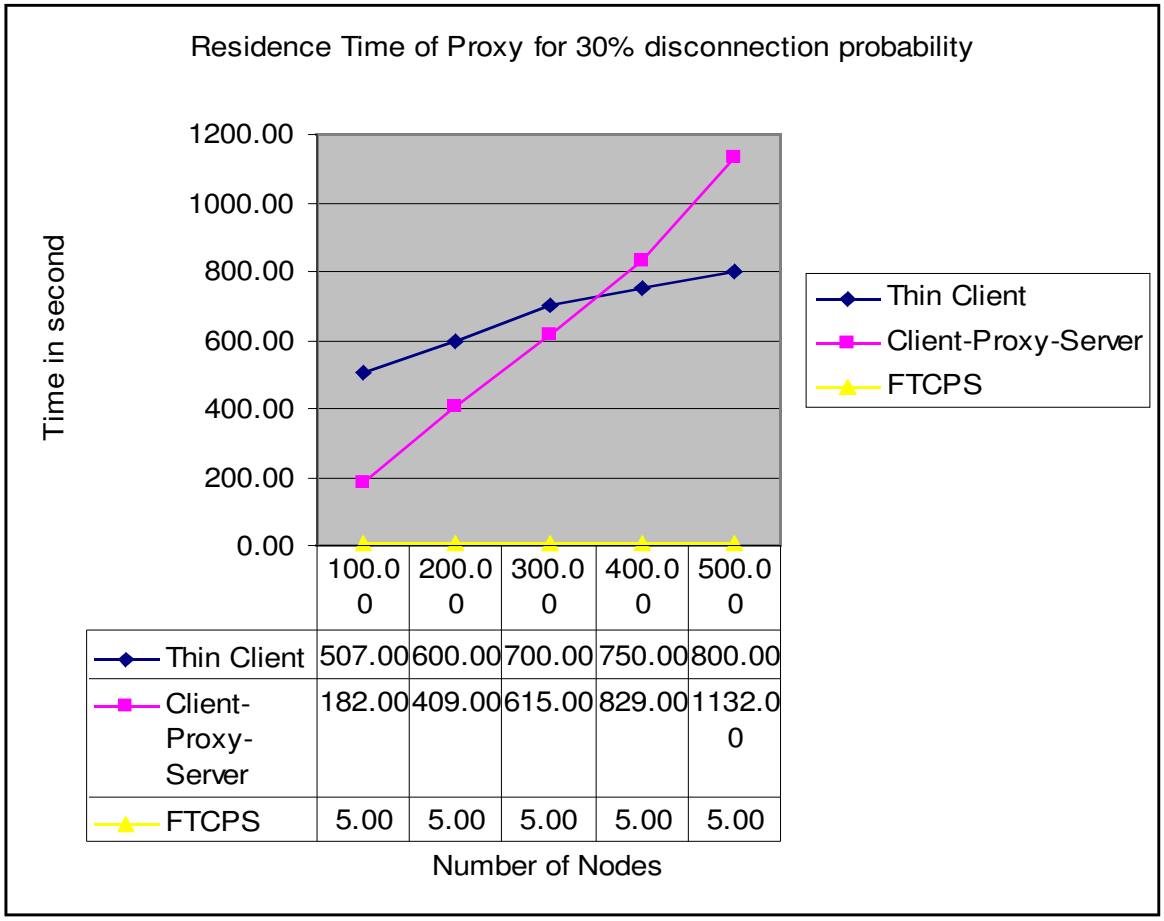


	100	200	300	400	500
Thin Client	485	495	500	505	508
Full Client	1158.00	2315	3541	4674	5864
Client-Proxy-Server	23.00	18	25	30	33
FTCPS	75.00	116	220	292	366

Figure 35: Residence Time of Client for 70% disconnection probability

-

Figure 50 shows the residence time of events at proxies for different architectures at 30% disconnection probability. The Full Client Architecture does not have a proxy. That is why we are comparing three architectures instead of four for residence time in Figures 50 through 53.

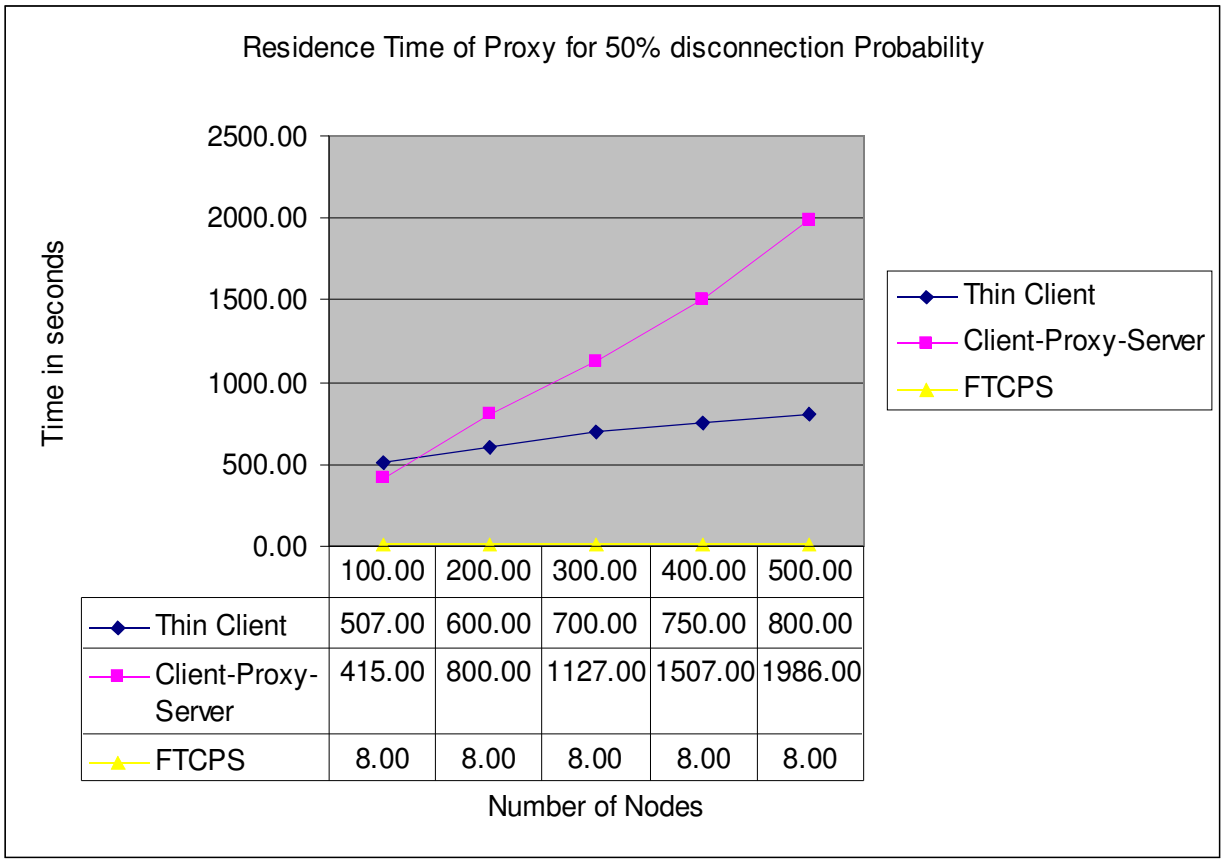


	100.00	200.00	300.00	400.00	500.00
Thin Client	507.00	600.00	700.00	750.00	800.00
Client-Proxy-Server	182.00	409.00	615.00	829.00	1132.00
FTCPS	5.00	5.00	5.00	5.00	5.00

Figure 36: Residence Time of Proxy for 30% disconnection probability

-

Figure 51 shows the residence time of events at proxies for different architectures at 50% disconnection probability. The difference between various architectures becomes evident as the number of nodes is increased and the disconnection probability is varied. FTCPS has a constant residence time which makes it the best choice for large mobile networks. FTCPS scales up very well from Figure 51.

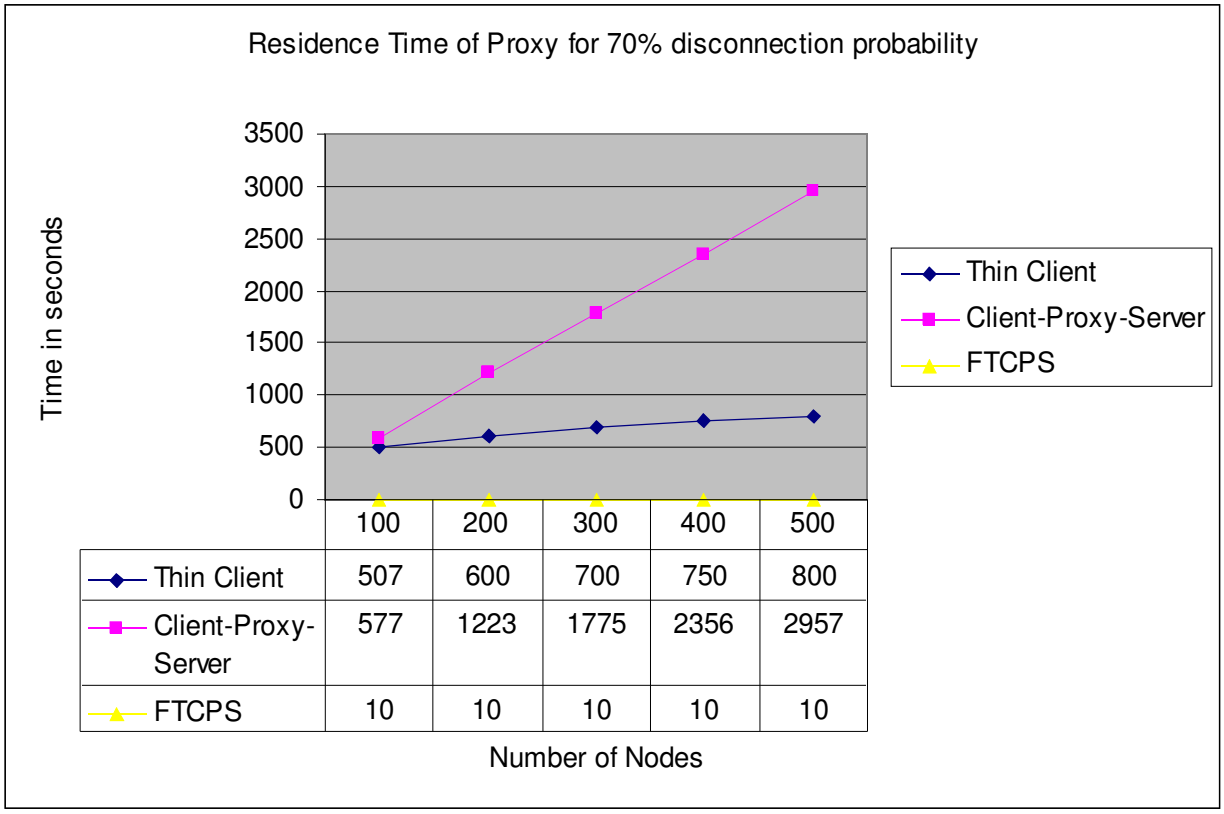


	100.00	200.00	300.00	400.00	500.00
Thin Client	507.00	600.00	700.00	750.00	800.00
Client-Proxy-Server	415.00	800.00	1127.00	1507.00	1986.00
FTCPS	8.00	8.00	8.00	8.00	8.00

Figure 37: Residence Time of Proxy for 50% disconnection probability

-

Figure 52 shows the residence time of events at proxies for different architectures at 70% disconnection probability.

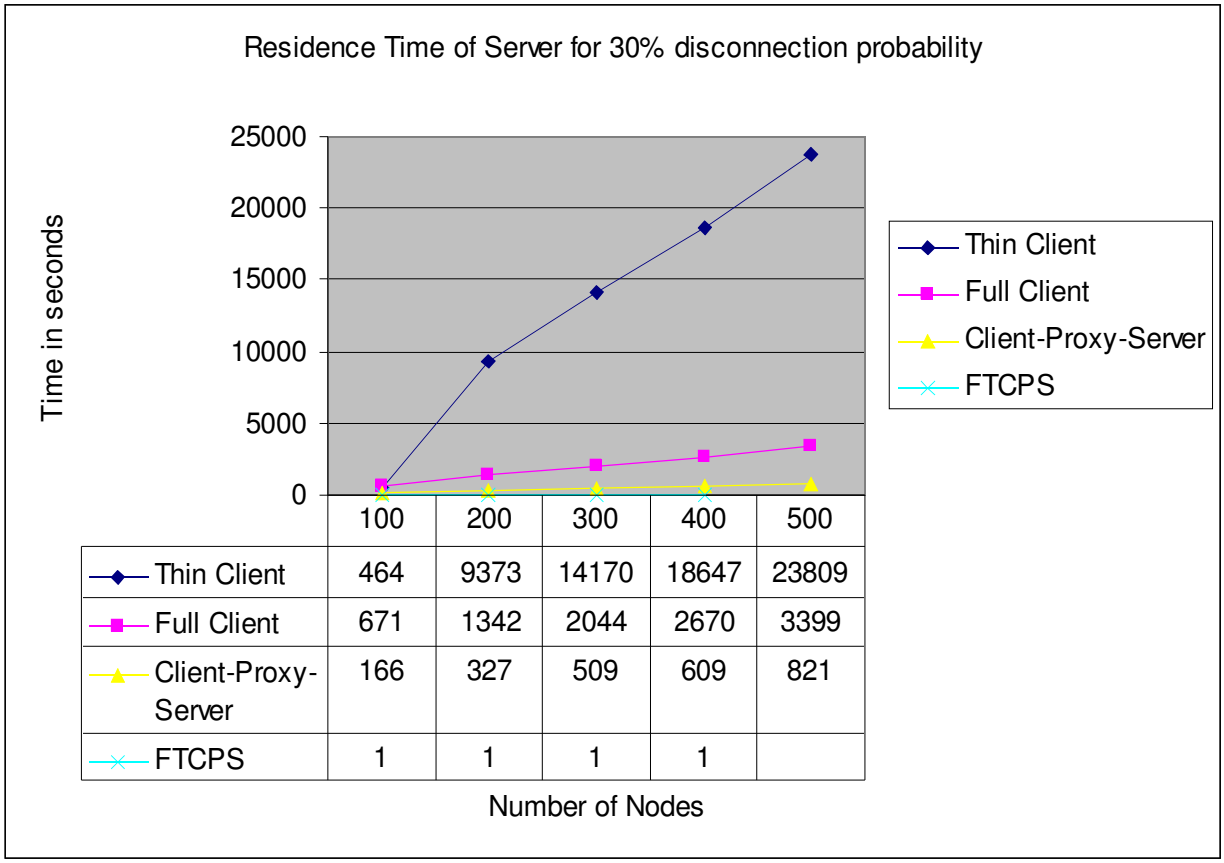


	100	200	300	400	500
Thin Client	507	600	700	750	800
Client-Proxy-Server	577	1223	1775	2356	2957
FTCPS	10	10	10	10	10

Figure 38: Residence Time of Proxy for 70% disconnection probability

-

Figure 53 shows the residence time of events at server for different architectures at 30% disconnection probability. The residence time of server in FTCPS is constant .

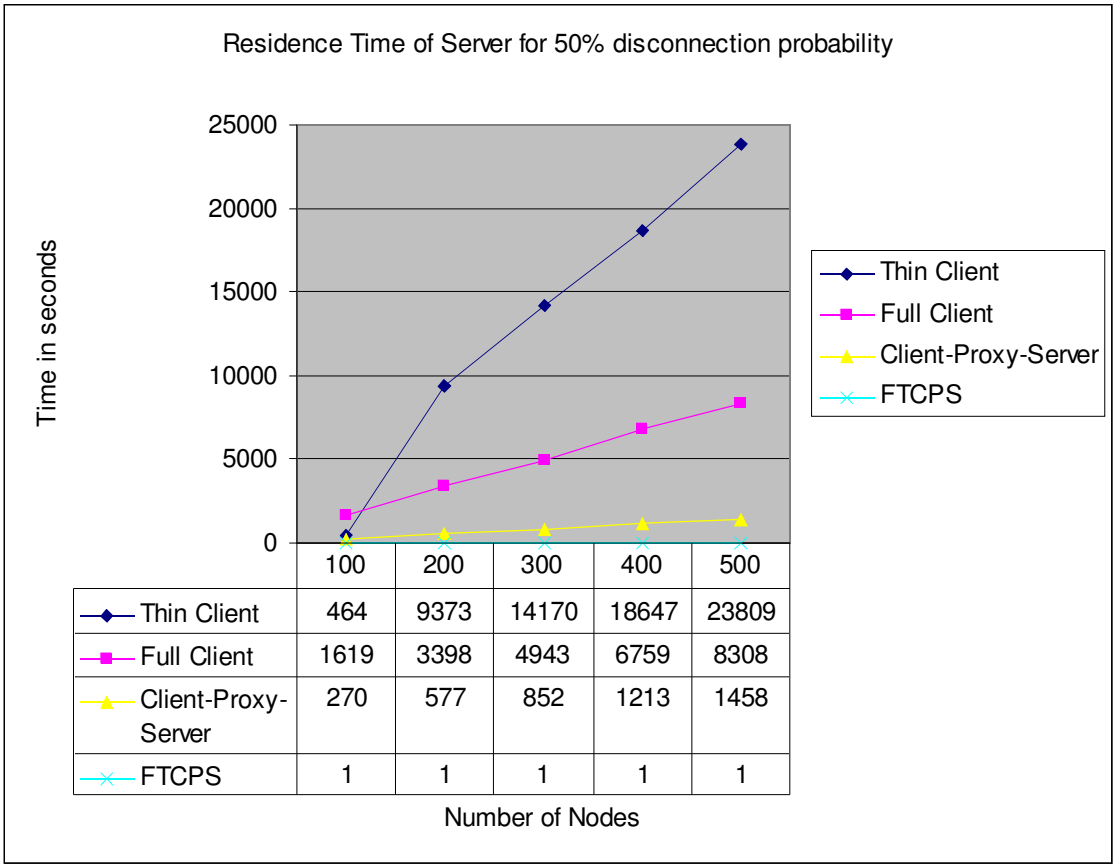


	100	200	300	400	500
Thin Client	464	9373	14170	18647	23809
Full Client	671	1342	2044	2670	3399
Client-Proxy-Server	166	327	509	609	821
FTCPS	1	1	1	1	

Figure 39: Residence Time of Server for 30% disconnection probability

-

Figure 54 shows the residence time of events at server for different architectures at 50% disconnection probability. The residence time of server in FTCPS is constant.

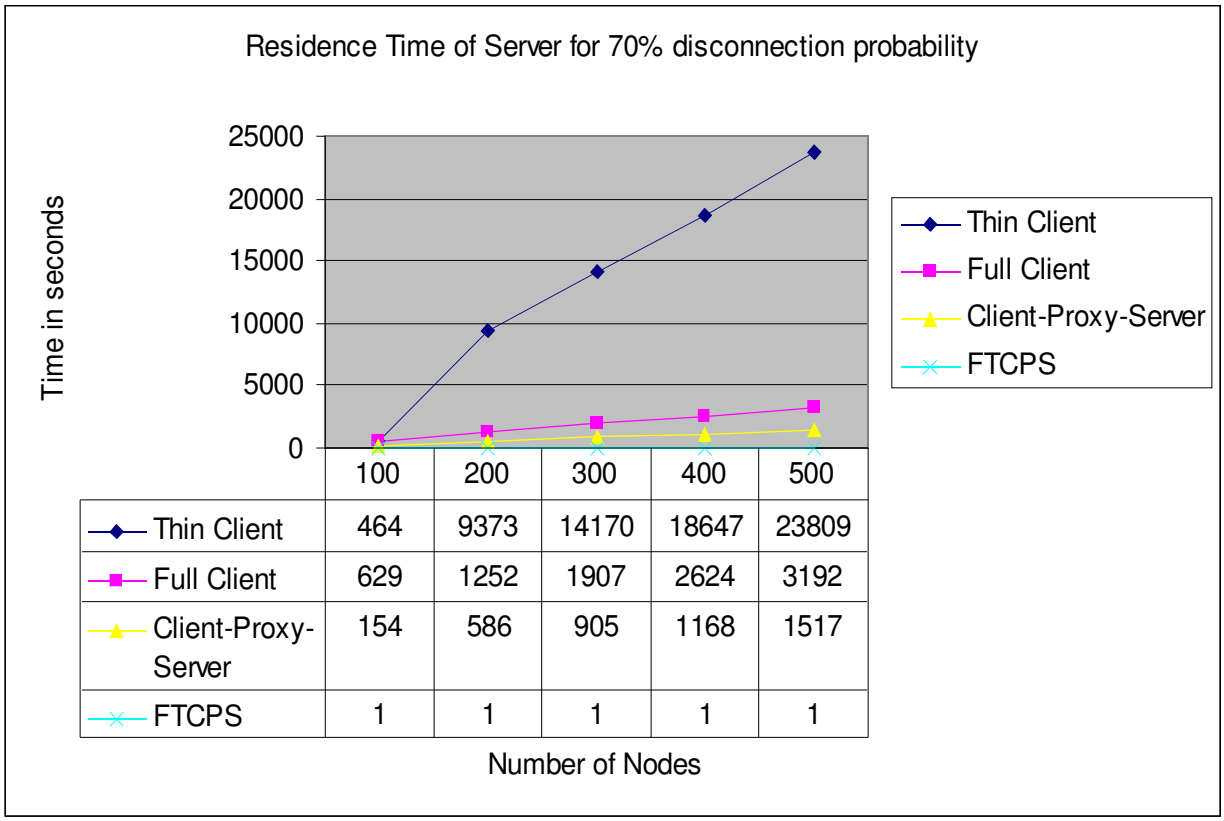


	100	200	300	400	500
Thin Client	464	9373	14170	18647	23809
Full Client	1619	3398	4943	6759	8308
Client-Proxy-Server	270	577	852	1213	1458
FTCPS	1	1	1	1	1

Figure 40: Residence Time of Server for 50% disconnection probability

-

Figure 55 shows the residence time of events at server for different architectures at 70% disconnection probability. The residence time of server in FTCPS is constant.

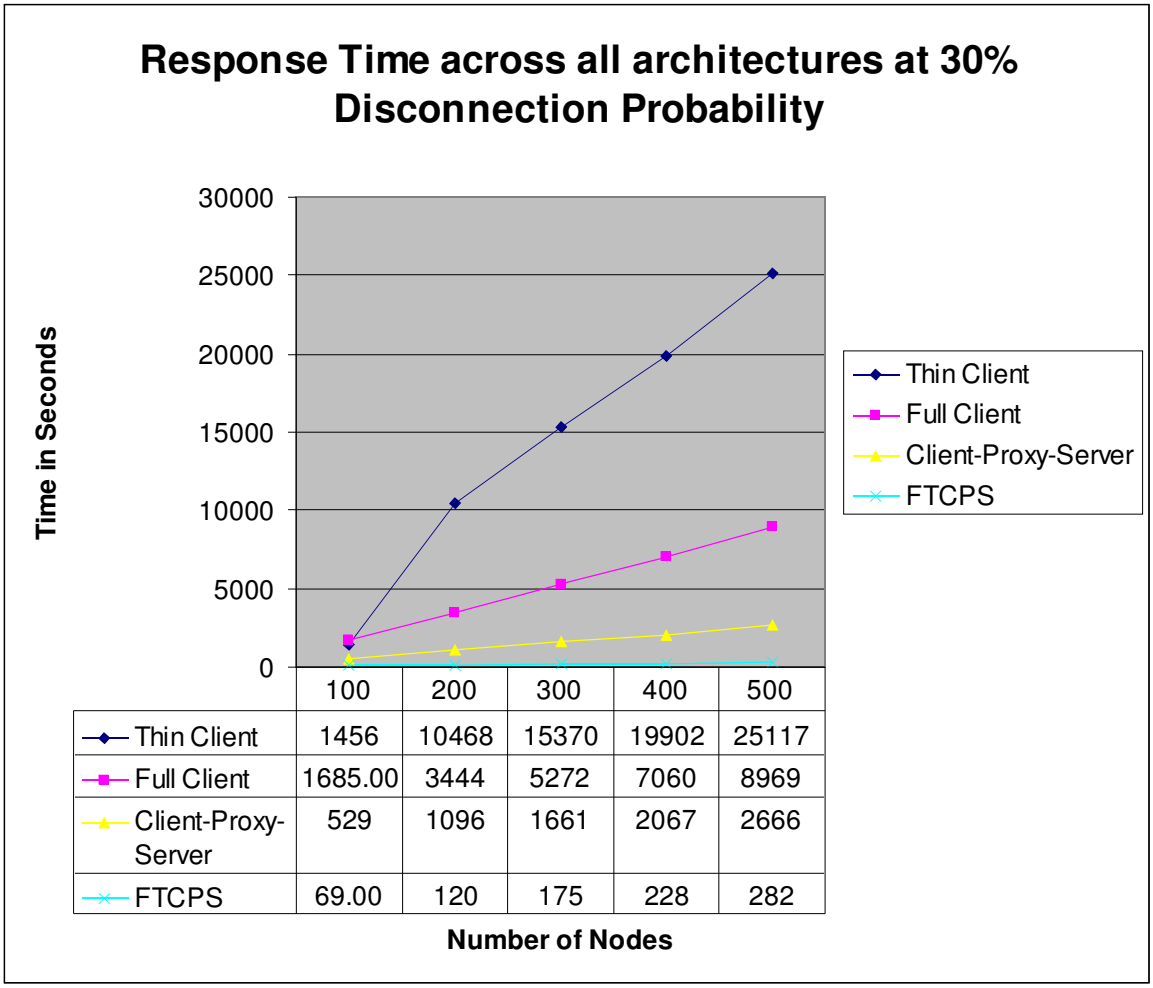


	100	200	300	400	500
Thin Client	464	9373	14170	18647	23809
Full Client	629	1252	1907	2624	3192
Client-Proxy-Server	154	586	905	1168	1517
FTCPS	1	1	1	1	1

Figure 41: Residence Time of Server for 70% disconnection probability

-

Figure 56 shows the response time (event-turn-around-time) for different architectures at 30% disconnection probability. The residence time of client, proxy and server makes up for the response time or event turn around time i.e the time taken for a request generated at a mobile unit to get its response back. The response time is the least in FTCPS.

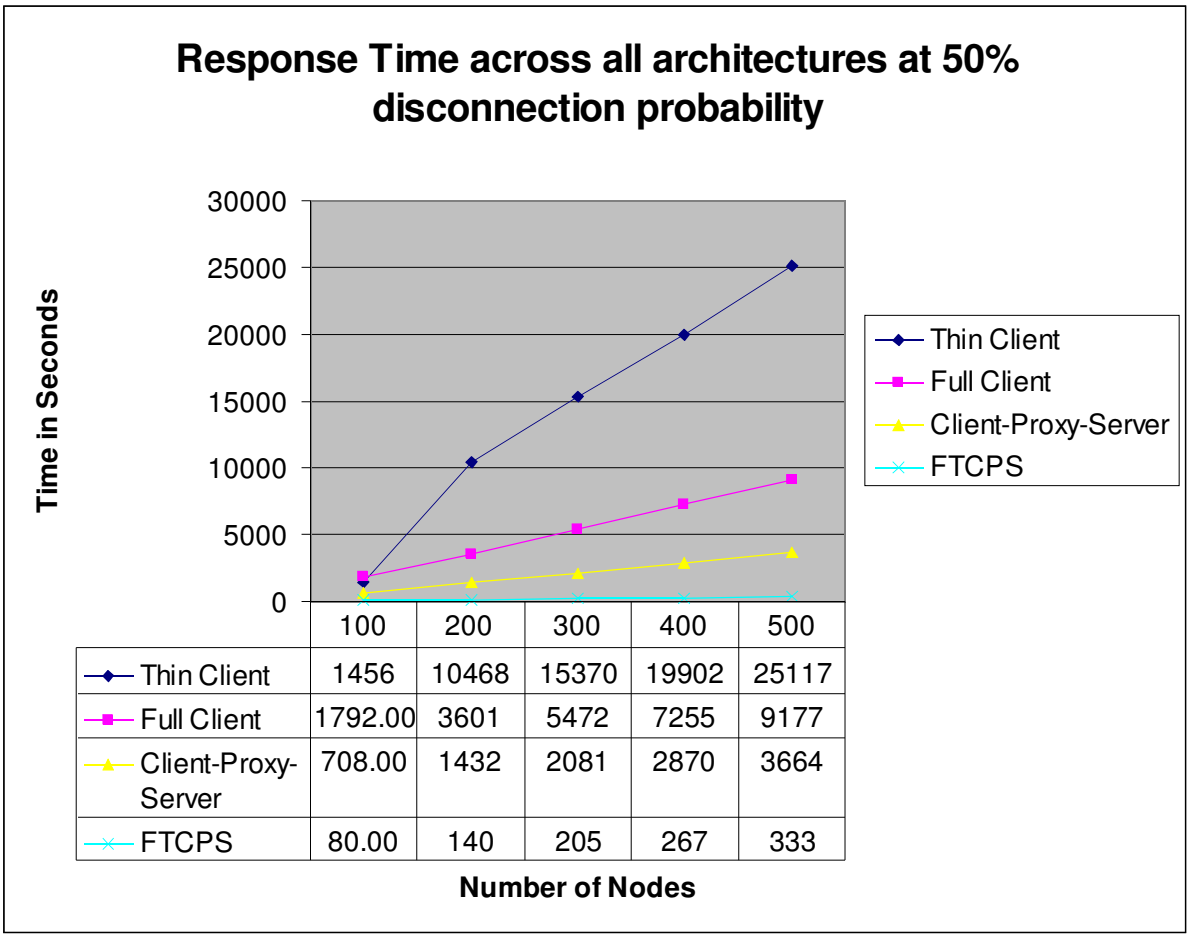


	100	200	300	400	500
Thin Client	1456	10468	15370	19902	25117
Full Client	1685.00	3444	5272	7060	8969
Client-Proxy-Server	529	1096	1661	2067	2666
FTCPS	69.00	120	175	228	282

Figure 42: Response Time across all architectures at 30% disconnection probability

-

Figure 57 shows the response time (event-turn-around-time) for different architectures at 50% disconnection probability. The response time in thin client architecture is very high and makes it less scalable as the server becomes the bottleneck of the system.

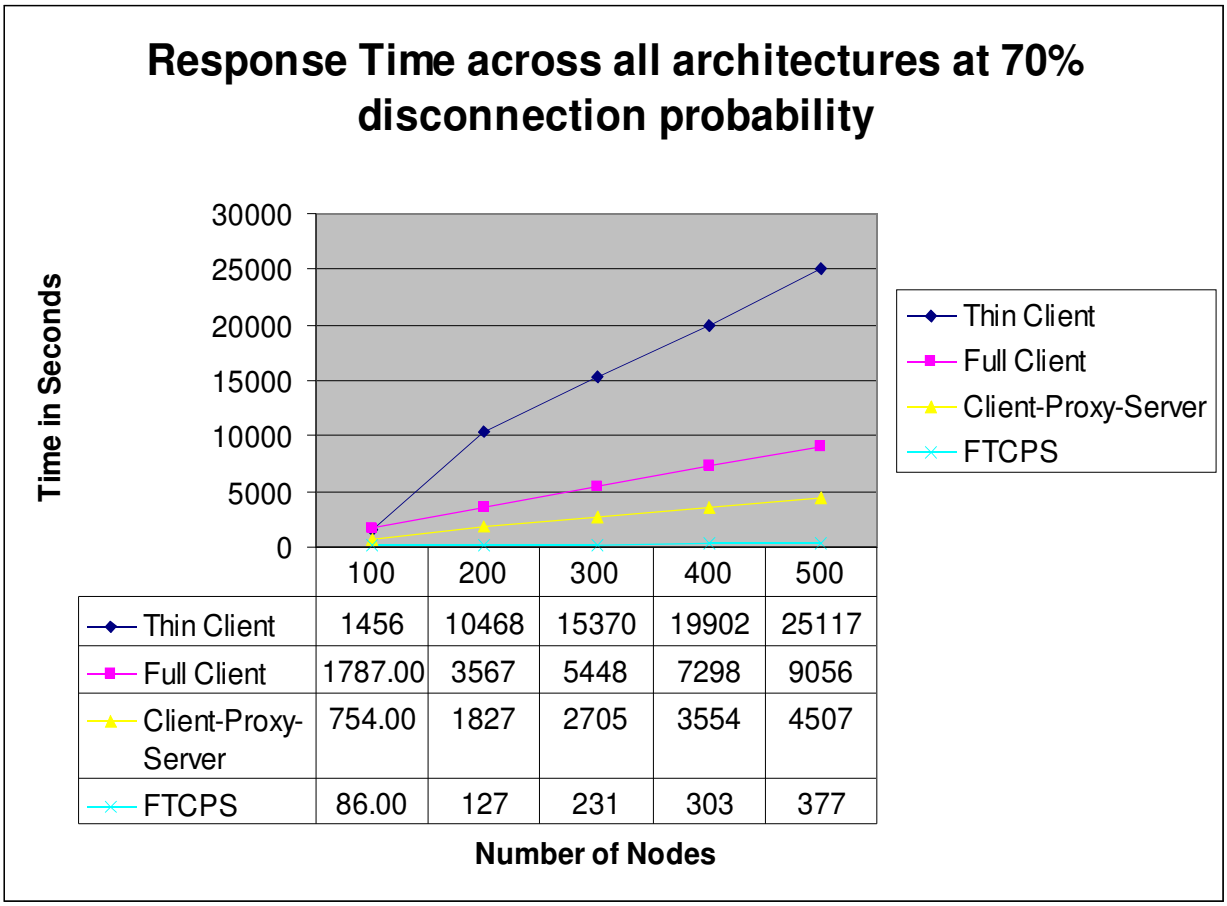


	100	200	300	400	500
Thin Client	1456	10468	15370	19902	25117
Full Client	1792.00	3601	5472	7255	9177
Client-Proxy-Server	708.00	1432	2081	2870	3664
FTCPS	80.00	140	205	267	333

Figure 43: Response Time across all architectures at 50% disconnection probability

-

Figure 58 shows the response time (event-turn-around-time) for different architectures at 70% disconnection probability. The response time of FTCPS is best across any disconnection probability and even with the increased number of nodes in the network.



	100	200	300	400	500
Thin Client	1456	10468	15370	19902	25117
Full Client	1787.00	3567	5448	7298	9056
Client-Proxy-Server	754.00	1827	2705	3554	4507
FTCPS	86.00	127	231	303	377

Figure 44: Response Time across all architectures at 70% disconnection probability

6.1 Concluding Remarks

We have examined the performance of four architectures for their residence times. The thin client architecture is not scalable as the server became the bottleneck for performance. The full client architecture is not suitable for mobile networks as the architecture cannot tolerate disconnections. The client-proxy-server architecture is a traditional approach where the proxies handle the requests only under disconnections. This architecture failed to scale well when the probability of disconnections is high and behaved closely like thin-client architecture under high disconnections. Our contribution of this thesis is to measure the "scalability" factor of a system from metrics like event residence time and response time (event-turn-around-time).

Another contribution of our work is the proposal for a new proxy architecture which envisions mobile devices to be servers. This architecture suits well for mobile Peer-to-Peer networks where the disconnections are needed to be tolerated gracefully. Our empirical study shows that FTCPS tolerated the disconnections gracefully and the response time never shot up even when the number of nodes and clusters were increased in our system.

6.2. Future Work

This thesis has prepared a test-bed for testing various architectures for their fault-tolerance, scalability and response time issues. It is a good idea to come up with a frame-work to extend the existing simulated models to test various applications. Synchronization policies of mobile unit and proxy are potential areas on their own. Also, various caching policies to reduce the latency in a system can be studied. The existing architectures can be studied under events which follow various mathematical distributions. Systems can be examined for their optimal performance by varying the processing times, probabilities of disconnection, mathematical distributions followed by the events across various architectures in the simulation. Further research can be done for answering questions like,

-

How does a proxy authenticate itself in a network?

How can a proxy be used to create new services based on the location and mobility information?

Should the proxy be centralized or distributed?

How should the proxy be deployed?

How should proxy be used to maintain user and device profiles to offer better services?

-

7. REFERENCES

[1]Adya, A., Bahl, P., and Qiu, L. Analyzing the browse patterns of mobile clients. In Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement (2001), ACM Press, pp. 189–194.

[2]Barrett, R., Maglio, P. P., and Kellem, D. C. Wbi: a confederation of agents that personalize the web. In Proceedings of the first international conference on Autonomous agents (1997), ACM Press, pp. 496–499.

[3]Crovella, M. E., and Bestavros, A. Self-similarity in World Wide Web traffic: evidence and possible causes. IEEE/ACM Transactions on Networking 5, 6 (1997), 835–846.

[4] Duska, B. M., Marwood, D., and Freeley, M. J. The measured access characteristics of World-Wide-Web client proxy caches. In Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97) (Monterey, CA, 1997).

[5] Forman, G. H., and Zahorjan, J. The challenges of mobile computing. Tech. Rep. TR-93-11-03, 1993.

[6] Gerald, A. E.-P. Role(s) of a proxy in location based services.

[7] Gribble, S. D., and Brewer, E. A. System design issues for Internet middleware services: Deductions from a large client trace. In Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS- 97) (Monterey, CA, 1997).

[8] Gupta, A., and Baehr, G. Ad insertion at proxies to improve cache hit rates.

[9] Hadjiefthymiades, S., and Merakos, L. Proxies + path prediction: improving web service provision in wireless-mobile communications. Mob. Netw. Appl. 8, 4 (2003), 389–399.

[10] Jing, J., Helal, A. S., and Elmagarmid, A. Client-server computing in mobile environments. ACM Comput. Surv. 31, 2 (1999), 117–157.

[11] Joshi, A. On proxy agents, mobility, and web access. Mob. Netw. Appl. 5, 4 (2000), 233–241.

[12] Junseok Hwang, P. A. Proxy-based middleware services for peer-to-peer computing in virtually clustered wireless grid networks.

[13] Kato, T., Ishikawa, N., Sumino, H., Hjelm, J., Yu, Y., and Murakami, S. A platform and applications for mobile peer to peer communications.

[14] Kuenning, G. H., and Popek, G. J. Automated hoarding for mobile computers. In Proceedings of the sixteenth ACM symposium on Operating systems principles (1997), ACM Press, pp. 264–275.

[15] Kunz, T., Barry, T., Black, J. P., and Mahoney, H. M. Wap traffic: description and comparison to www traffic. In Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems (2000), ACM Press, pp. 11–19.

[16] Lou, W., and Lu, H. Efficient prediction of web accesses on a proxy server. In Proceedings of the eleventh international conference on Information and knowledge management (2002), ACM Press, pp. 169–176.

[17] Markatos, E. P., Pnevmatikatos, D. N., Flouris, M. D., and Katevenis, M. G. H. Web-conscious storage management for web proxies. IEEE/ACM Trans. Netw. 10, 6 (2002), 735–748.

-

[18] Mummert, L. B., Ebling, M. R., and Satyanarayanan, M. Exploiting weak connectivity for mobile file access. In Proceedings of the fifteenth ACM symposium on Operating systems principles (1995), ACM Press, pp. 143–155.

[19] Padmanabhan, V. N., and Qiu, L. The content and access dynamics of a busy web site: findings and implications. In Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (2000), ACM Press, pp. 111–123.

[20] Rabinovich, M., Chase, J., and Gadde, S. Not all hits are created equal: cooperative proxy caching over a wide-area network. Computer Networks and ISDN Systems 30, 22–23 (1998), 2253–2259.

[21] Rao, C.-H. H., Chen, Y.-F. R., Chang, D.-F., and Chen, M.-F. imobile: a proxy-based platform for mobile services. In Proceedings of the first workshop on Wireless mobile internet (2001), ACM Press, pp. 3–10.

[22] Singh, A., Trivedi, A., Ramamritham, K., and Shenoy, P. Ptc : Proxies that transcode and cache in heterogeneous web client environments.

[23] Wang, J. A survey of web caching schemes for the internet. SIGCOMM Comput. Commun. Rev. 29, 5 (1999), 36–46. 17

[24] Zenel, B., and Duchamp, D. A general purpose proxy filtering mechanism applied to the mobile environment. In Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking (1997), ACM Press, pp. 248–259. 18

[25] A tutorial on SIMJAVA http://www.icsa.informatics.ed.ac.uk/research/groups/hase/SIMJAVA/

[26] Sushil K. Prasad, Vijay Madisetti, Shamkant B. Navathe, Raj Sunderraman, Erdogan Dogdu, Anu Bourgeois, Michael Weeks, Bing Liu, Janaka Balasooriya, Arthi Hariharan, Wanxia Xie, Praveen Madiraju, Srilaxmi Malladi, Raghupathy Sivakumar, Alex Zelikovsky, Yanqing Zhang, Yi Pan, and Saied Belkasim. SyD: A Middleware Testbed for Collaborative Applications over Small Heterogeneous Devices and Data Stores,, In *Proceedings of ACM/IFIP/USENIX, 5th International Middleware Conference*, Toronto , Ontario , Canada , October 18th - 22nd, 2004