Service Recommendation in SLP

by

David McCormack

205075

An honours project submitted to the
School of Computer Science
in partial fulfillment of the requirements
for a Bachelor of Computer Science

Prepared for:

M. Barbeau

Carleton University
School of Computer Science

F. Bordeleau

Carleton University
School of Computer Science

August 18, 2000

Abstract

The Service Location Protocol (SLP) allows for service discovery. Within SLP, mechanism exist that allows for services to query based on the attributes of that services. This report details the extension of SLP to include mechanisms for service recommendation.

Service recommendation allows UAs to discover one SA which they consider optimum. This was performed by extending the predicate operators to include ones which enable UAs to specify how all the services should be compared, and the criteria used for determining an SA as optimal.

These contributions were implemented in the context of a proxy server and web browser. In several simulations, they were tested and proven to offer a significant improvement in reducing delay times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past five years the web has exploded in popularity. Initially starting out in the academic world it has spread through the commercial and residential worlds, becoming an integral part of our lives. Many people depend on the web for retrieval of information more than any other source. In this new information-based reality, any disruption or abruption in web services can no longer be accepted.

There are numerous organizations who utilize proxy servers as their means for accessing the internet. These servers act as central conduits to co-ordinate access to external information from within the organization as explained in Section 1.1.2.

Unfortunately, since proxy servers are a centralized conduit, they can easily experience saturation. A means of automatically selecting and reselecting a proxy servers is the motivation for the extension purposed and implemented in this report.

A service discovery method such as Service Location Protocol, explained in depth in the Introduction (Section 1.1.1), provides transparent facilities for the discovery of services. Unfortunately, this protocol does not have means for advising users on the service which best suits their needs.

Service recommendation is the evaluation of all available services, and the subsequent recommendation of the single service which best satisfies the user's needs. Section 2 contains a definition of service recommendation along with how SLP can be extended to perform it.

Section 3 on an application of service recommendation, discusses the issues which must be considered when creating an environment which allows for service recommendation. Here, all of the issues of service recommendation are examined from the perspective of a proxy server.

The specifics of the implementation of service recommendation into a proxy server is contained in Section

4.

The evaluation of the performance of service recommendation in the context of the proxy server application is examined in Section 5.

1.1 Background

This section contains a brief explanation of Service Location Protocol followed by an explanation on proxy servers and web browsers.

1.1.1 Service Location Protocol

The Service Location Protocol version 2 (SLP), defined in RFC2608 [Guttman et al., 1999b], is a method for the discovery of services. The protocol has two main modes of operation, both of which are described below.

The first mode is typical bare bones operations where a user, User Agent (UA), discovers a service. The UA transmits a service requests by broadcast. The service providers, known as Service Agent (SAs), listen for service requests and compose a reply in the form of a service registration if they provide a service which the UA is searching for.

The second mode of operations is when a Directory Agent (DA) is deployed. A DA is an entity which keeps track of all the services on a network. When a UA is searching for SAs, it sends out a service request. This message is sent by unicast to the DA, which causes the DA to compose a reply with all the SAs which satisfy this request.

Service Agent

A Service Agent (SA) provides services to users. The SA informs the network about their services by broadcasting or unicasting the service registration message to an SA or DA (if present). The service registration contains the URL of the service, the attributes of the service, and the scopes (administrative domains) to which it belongs.

All of the SAs listen for messages sent from DAs. A message which contain the address of the DA is then used for the sending of subsequent service registrations using Unicast.

User Agent

A User Agent (UA) is an entity which wishes to locate a service. When a UA is searching for a service, it composes a service request for a desired service containing: the type of the service desired, the scope(s) to which the UA belong, and a predicate expression of which service to select (optional). A predicate is an expression to which the service must conform to. For example, a predicate in a service request for the discovery of a printer would be: `(color=true)`. The UA receives only registrations from the SAs containing colour printers.

The UA listens for messages sent from DAs. A message which contain the address of the DA is then used for the sending of subsequent service requests using Unicast.

Directory Agent

The Directory Agent (DA), is an entity which holds information about the services on a network. It drastically reduces the number of messages transmitted since both service requests and service replies are unicast to the DA instead of being broadcast on the entire network.

Directory Agent Mesh

One of the problems with DAs is that the number of services registered with them is limited to the ones on their network. The algorithm, as presented in the draft [Zhao and Schulzrinne, 2000], provides facilities for directory agents to exchange knowledge of each others registered services. These communicating directory agents, when present, offer more services, hence increasing the performance for users. Figure 1.1, illustrates how the algorithm actually provides more services.

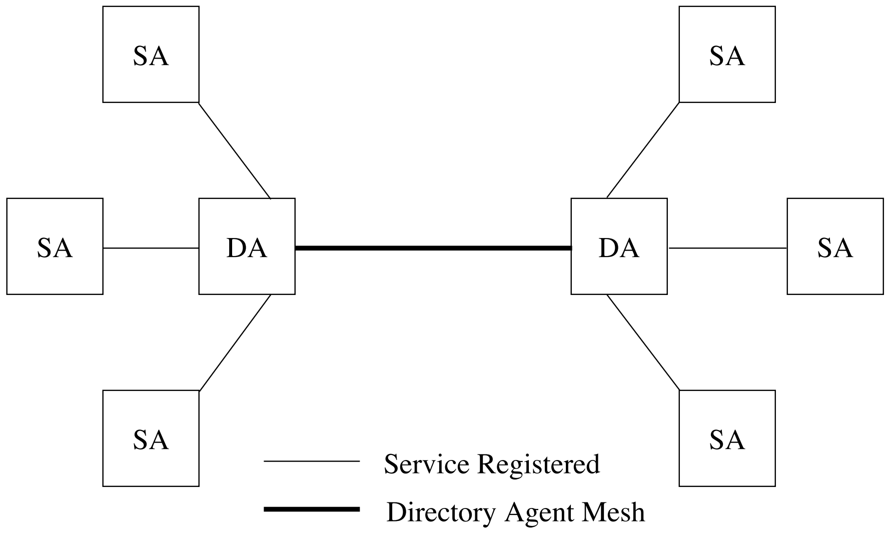

Figure 1.1: Topology of a network with the directory agent mesh algorithm deployed

Service Subscription

SAs who have variable attributes may update their attributes by performing an incremental registration which updates the attributes with the DA. These updated attributes are only known to one DA, and none of the UAs become aware of the change until they send out subsequent service requests.

This is a problem for those utilizing SLP for Quality of Service applications because a service constantly changing attributes increases complexity for users of the service who are not informed of the changes.

These issues have been addressed in a draft on Notification and Subscription [Kempf and Goldschmidt, 2000]. This paper introduces the notion of service subscription, which allow UAs to subscribe to services, meaning they will be informed when the service changes.

Templates

The properties of a service type are formally defined in service templates. The syntax of the templates are defined in RFC2609[Guttman et al., 1999a], and provide information on the attributes of the service. Each attribute defined in a service template consists of flags and a type. The flags denote whether the attribute is optional, multi value, literal, or required in service requests. The types of the attributes are either integer, boolean, opaque, or string. All of the templates are available at Service Template Repository[Tem, 2000].

1.1.2 Proxy Server

A proxy is someone who acts on your behalf. Similarly, a proxy server acts as an intermediary between clients (web browsers) who are using the proxy server. For example, if a client wishes to load a web page, their request travels from their browser to the proxy server. The proxy server then sends out the request onto the internet. When the response arrives, the data is forwarded to the client.

The notion of proxy servers seems to add another level of indirection for all web requests. But this level of indirection can actually reduce transmission times through caching. Many proxy servers keep copies of frequently accessed web pages locally in a cache. Each time a request is received, it is only sent out to the internet if it is not contained in the cache. This allows for a reduction on the number of requests for data set out onto the internet.

There are a number of proxy servers deployed: Microsoft Proxy Server [msp, 2000], Netscape Proxy Server [nsp, 2000], Novell BorderManager [nov, 2000], and Squid Proxy Cache [squ, 2000].

1.1.3 Web Browser

Web browsers allow users to view pages that have been published on the internet. Users navigate in a browser by loading and viewing different pages. When the user wishes to load a new page, it sends out a request to the host of the page asking for the content. The content is then delivered back to the browser which interprets and displays the data. There are many web browsers today, including: Microsoft Internet Explorer [msi, 2000], Netscape Navigator[net, 2000], and Mozilla [moz, 2000],

Chapter 2

Service Recommendation Operators

This section contains information on the main issue addressed in this report, that of service recommendation. It will begin with a clear definition of the problem. This is then followed by the context in which the problem occurs. The end of this section presents a solution to the service recommendation issues.

2.0.4 Problem Definition

The problem, precisely defined, is: how can an optimal service be determined based upon criteria provided by a UA.

2.1 Problem Context

The current functionality allows UAs to add predicates to their service requests. These predicates are evaluated for each service, and all the services who satisfy that request are returned.

The formulation of a predicate containing multiple variables into a single value is interesting due to the nature of the variables. Each variable's context of measurement is completely different, which presents problems when attempting combine them in a single equation (similar to trying to add apples and oranges).

2.2 Proposed Solution

This solution is centred around the notion of rank. Each service is ranked with respect to all other services. The one with the best rank then becomes the optimal service. This problem can be solved by the addition of operators for the evaluation of predicates. The first two operators, `optimallow()` and`optimalhigh()`, comprise the beginning of all predicates used when searching for an optimal service. These operators evaluate the predicate contained within them, and only return the single which has either the highest predicate value, or the lowest respectively.

The two operators above are sufficient for determining if a single attribute is optimal, but will not work for multiple attributes. In order to use multiple attributes together, they must be all on the same scale. This was achieved by using the rank of each variable. The two remaining operators required are: `ranklow()` and `rankhigh()`. These operators return the rank of the attribute among all the other services as an integer.

2.3 Solution Implementation

The modifications required to implement the operators were placed inside a Directory Agent (DA) in the predicate evaluator. The predicate evaluator used in SLP is the LDAP search filters described in RFC2254 [Howes, 1997]. The extensions proposed is that UAs now have the option of receiving a single response to a service request.

The optimal service equation is a predicate composed by a UA to reflect how they wish the optimal service to be determined. This predicate is then sent to the directory agent when searching for a service. The DA evaluates the predicate and returns the service which satisfies the criteria.

The process of determining the optimality of a service is difficult to characterize with a single predicate. The main reason for this is the different units and interpretation of the attributes. This issue was overcome by using the rank operator.

Two rank operators were implemented allowing for results to be returned for high and low valued items. The first operator, `ranklow(`**`var`**`)`, returns the placement of **`var`** relative to the lowest value variables with a value of 1 denoting that the SA contains the lowest value of all the variables contained for that service. Conversely, the `rankhigh(`**`var`**`)` returns the rank of **`var`** relative to the highest valued variable.

Consider the situation below, given in Table 2.1, where the UA sends the predicate `ranklow(Attr1) + rankhigh(Attr2)`.

URI	Attr1	Attr2	ranklow (Attr1)	rankhigh (Attr2)	ranklow(Attr1) + ranklow(Attr2)
service:proxy://star01	12	6000	2	4	6
service:proxy://star02	45	2000	4	2	6
service:proxy://star03	9	2000	1	2	3
service:proxy://star04	35	500	3	1	4

Table 2.1: Service Ranking Example

The results as shown in the last column of Table 2.1 illustrate how the rank operator can be used to generate a single service quality value for each service. The next step was to determine from all of the service quality values, which one should be selected. Two more operators were created, `optimallow(`**`x`**`)` and `optimalhigh(`**`x`**`)`. These work similarly to rank operators, by returning the single service from predicate **`x`** which is either the lowest or highest value respectively.

The above solution functions worked, but did not provide facilities for weighing the importance of the attributes. This was done through the `*` operator, allowing the resulting data to be weighed differently. For example, if a UA is attempting to determine the optimal service, and the amount of UAs using the SA is unimportant, but the available bandwidth is, a predicate such as: `optimallow(ranklow(users) * 20 + rankhigh(bandwidth) * 80)` would be utilized as show in Table 2.2:

URI	Load	Bandwidth	rankhigh(bandwidth) * 20 + rankload(load) * 80	Optimal Service
service:proxy://star01	25	980	22	no
service:proxy://star02	5	1280	26	no
service:proxy://star03	10	412	12	yes
service:proxy://star04	30	8521	40	no

Table 2.2: Sample Proxy Selection Results

The solution presented does contain a limitation arising from the use of the rank operator. The rank operator does not provide any information on the differences between the ranks. So, for example, in the data set $\{1, 100, 101, 104, 106, 110\}$, the difference between the items with rank 1 and 2 is very significant, yet this is not reflected in the results returned by the rank operator.

Chapter 3

Application of Service Recommendation

We examine the performance of service recommendation through the use of a proxy server. This was selected due to the simplistic nature of proxy servers and web browsers. In this context, each proxy server is an SA and each web browser is a UA. The application of the service recommendation is the selection of the optimal SA for each UA.

The problem, upon examination, has three possible solutions. Each of these possible solutions are examined and then compared. Before presenting each possible solution, the limits of the solutions are discussed.

3.1 Solution Constraints

There are several constraints to which every solution must comply to. The following subsections denote these reasons and the necessity of compliance to them.

3.1.1 Service Discovery through Service Location Protocol

The discovery of the proxy services is to be done through the Service Location Protocol as defined in RFC 2608 [Guttman et al., 1999b], which allows for the dynamic discovery of services.

There exist other service discovery protocols other than the Service Location Protocol which provide similar functionality. One is the Light Weight Directory Access Protocol [Wahl et al., 1997], which was not selected because it requires the users to enter the directory address. Another possible solution is the use of JINI [Jin, 1998], but this is inadequate since it is dependent on the Java virtual machine.

3.1.2 Directory Agents Deployment

The service discovery mechanism, as discussed above, is required to be implemented as a directory agent. This requirement is necessary since they provide substantial network savings, which are required for scalability to a large scale deployment of user and service agents.

3.1.3 Meshing Directory Agent Considerations

Any possible solution must be compatible with the meshing directory agent algorithm. Although it is not widespread in usage, it will most likely become part of all standard SLP implementations. This mesh will increase performance for end users by providing them with access to all the available proxy servers, meaning that any design solution should be conscious of this functionality.

3.1.4 Service Subscription

Unfortunately, the SLP implementation of use does not support service subscription as described in Section 1.1.1. In the context of the problem, if service subscription was implemented, it would allow for immediate selection of an optimal service. Since no implementation exists, the proposed solution is to have frequent re-registrations. Also, during solution design, service subscription issues should be considered, ensuring future performance benefits once implemented.

3.2 User Agent Based Solution

The UA based solution is one where the web browser is responsible for determining the best proxy choice. This solution does not require any modification to any part of the existing SLP framework. It does require that extensive modifications be placed into the web browser. At first this solution seems favorable since there is no impact on the SLP implementation. Upon closer consideration it does not scale well to large networks because the reply to each query would contain all the proxy servers known to the directory agent. It also does not promote re-use since every application that this selection algorithm is required to include the code in their application. Furthermore, if the operation of this solution is considered in a directory agent mesh, the network usage would be large since all proxy server and their attributes must be sent to the web browser.

In the message sequence chart depicted in Figure 3.1, we see that every service update from any subscribed service will send a message to the user agent since the user agent requires all information to determine the optimal service. These repeated attempts by the user agent to find the optimum proxy server by comparing the attributes of the current proxy server against the latest ones gathered will result in an enormous number of messages.

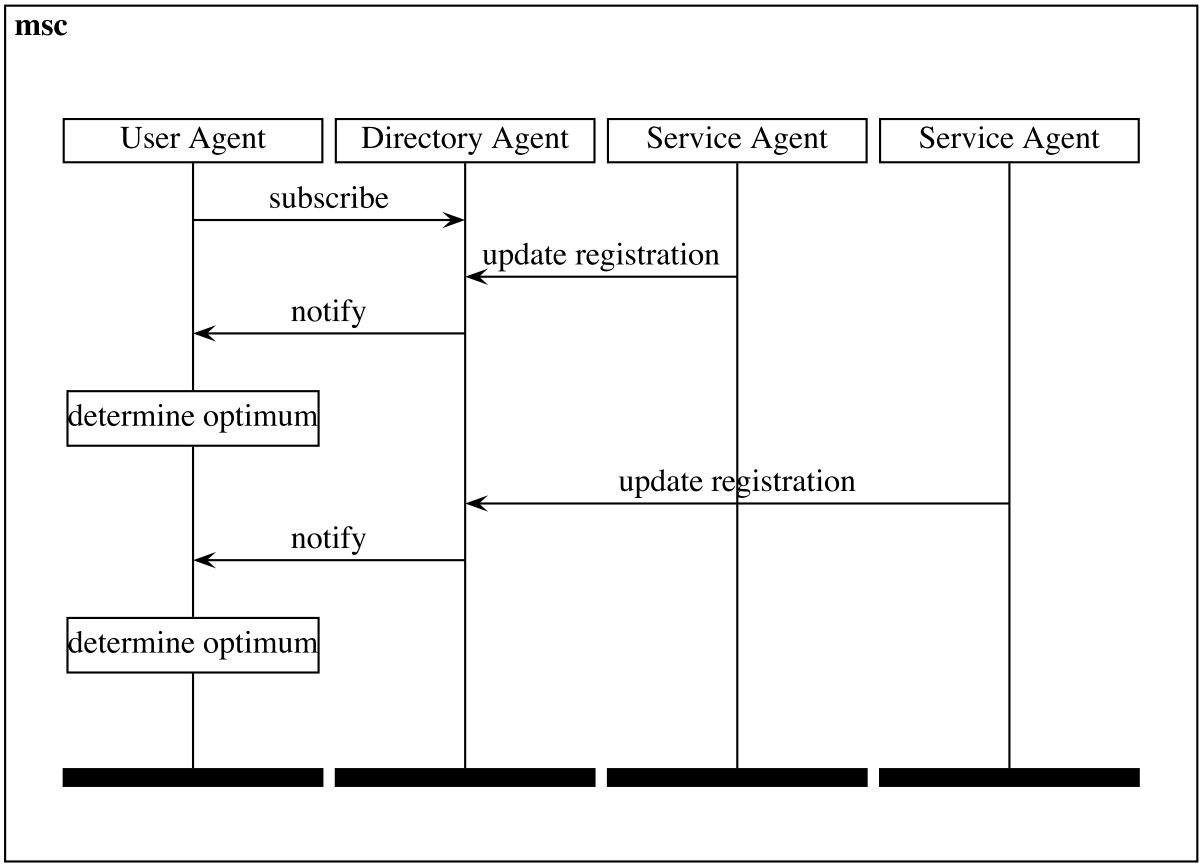

Figure 3.1: User agent based proxy selection

3.3 Service Agent Based Solution

A SA based solution would be very impractical. In this solution, the SA (proxy server) would be responsible for informing the UA (web browsers) of the service. This would work very well if there was only one proxy server and many web browsers. As the number of proxy servers increase, so does the complexity of a SA based solution, since each proxy server would have to determine if it is the optimal server among all the servers, and then inform the web browsers of this. This is a poor solution because it is redundant.

3.4 Directory Agent Based Solution

The DA based solution is one where the DA is responsible for informing the UAs of the optimum service agent for their needs. Each SA is required to inform the DA when its attributes change. The UA then receives the information from the DA about the SA attributes. This solution may appear at first to be bandwidth intensive, since the message travels first from the SA to the DA, and then to the UA. But in an environment with a DA deployed, this would happen anyway. This does have the added benefit of potentially reducing the number of messages since the DA is responsible for determining if the UAs should switch services, hence reducing the number of messages (since not every change in the proxy attributes will require the user agent to change services). As depicted in Figure 3.2, not all service update messages must be forwarded to all UAs. This is illustrated when the first service updates its attributes through service registration; the UA is only informed if it is considered an optimal service.

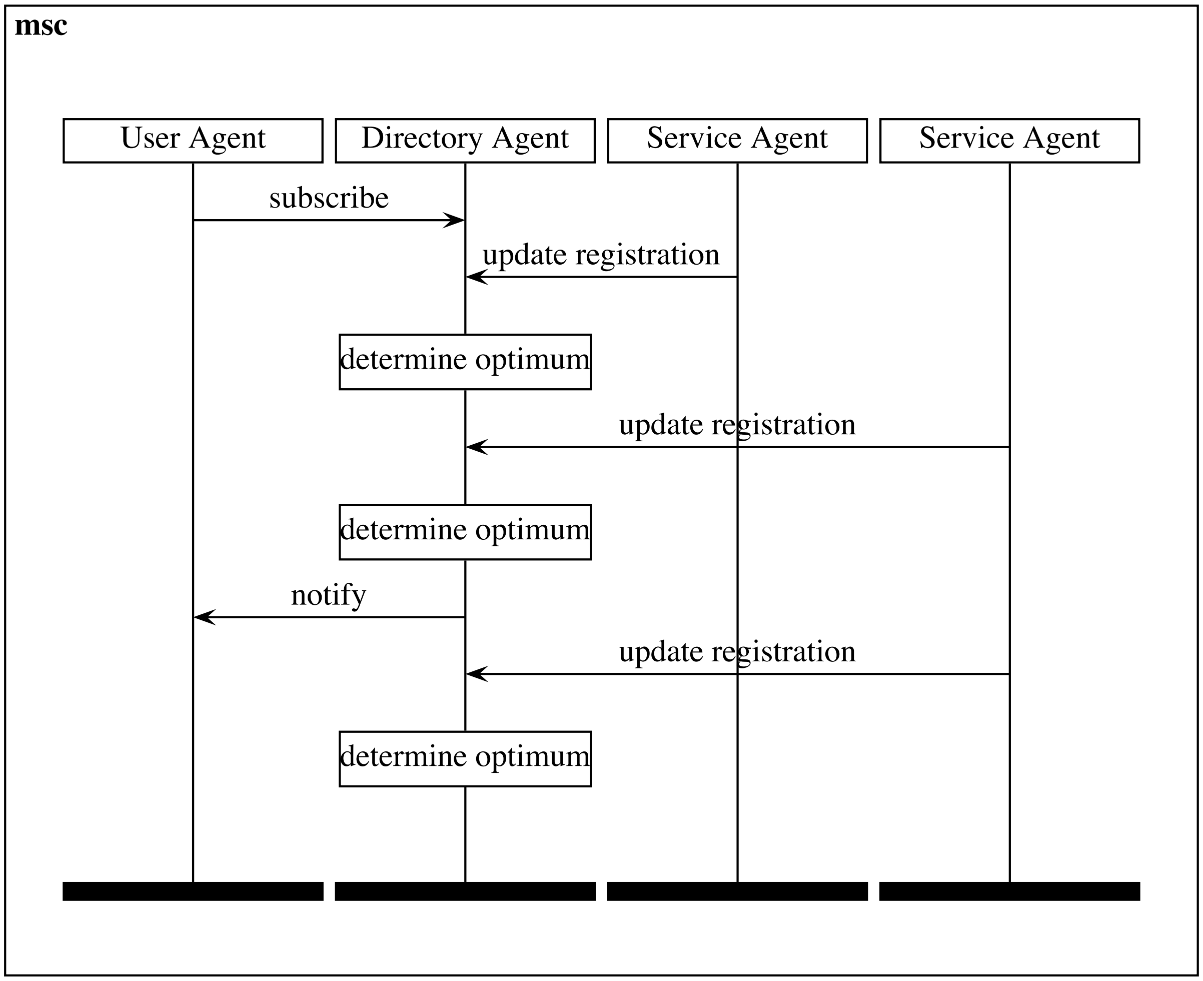

Figure 3.2: Directory agent based proxy selection

3.5 Solution Selection

The DA approach was selected because it has a minimal number of messages and the smallest impact on the UAs and SAs. This approach requires that the DAs support the service recommendation operations as listed in Section 2.2.

Chapter 4

Application Implementation and Extension

This contains information on how the DA based solution was implemented. The implementation is first discussed at a high level. The following section discusses the modifications of the SA, DA, and UA in depth.

4.1 Implementation Overview

The SAs, when registering, are required to register attributes which will be used to judge the quality of the service. These attributes can be used in the formation of a predicate. This predicate is an equation which is used to determine the optimal SA is for a UA. For example, a UA may consider the optimal service to be one containing the maximum available bandwidth and the minimum number of users.

4.2 Service Agent Modifications

The SA selected for the proxy server was squid proxy and caching program[squ, 2000]. It was selected because of its bountiful features, widespread usage, and source code availability.

The modification of squid, enabling SLP support, was straight forward except for attribute creation. Although the attributes string creation was simplified due to the Service Location Protocol Attribute Application Programmers Interface Extensions[Hughes, 2000], many difficulties arose in the location of the relevant usage and statistical information in the massive amount of code comprising squid. Once the location of the information was found, the modifications were complete expediently.

A nice feature of squid is the ability to generate many usage statistics in a per minute or per hour interval. This statistical data allowed for the several meaningful attributes be registered. These attributes, some are featured in Table 4.1, allow for the determination of service quality. The specific attributes were selected because of their ability to accurately reflect the performance parameters important to users, hence allowing for predictions on future service performance to be made.

Attribute Name	Attribute Description
`available-proxy-services`	This is a listing of all the potential proxy services, which typically include www and ftp.
`average-five-minute-kb-in-per-sec`	The average kilobytes/second transfered into the proxy server in five minutes.
`average-five-minute-kb-out-per-sec`	The average kilobytes/second transfered out of the proxy server in five minutes.
`cpu-usage`	This is the average CPU usage of the proxy server. It is a value in the range $[0, 100]$, with 0 being no usage, and 100 full usage.
`http-hits-per-second`	The average hits that the proxy server receives per second.
`http-request-per-second`	The average request per second that the server receives.
`number-of-clients`	The amount of currently connected clients to the proxy server.

Table 4.1: A Subset of the Proxy Server Advertised Attributes

The selection of an interval for the usage statistics was a difficult decision to make, due conflicting goals. One goal is having the most up-to-date information available to the DAs, while the other is maximizing the number of samples used in the calculation of averages. The solution to this issue was to create three ranges, 1 minute, 5 minute, and 1 hour, allowing user agents to evaluate the services performance. These intervals were used for the input and output speeds. Other attributes such as hit averages and request averages are advertised only in 1 range because squid only provides this data in a per second range. This per second data could be transformed into the above ranges, but was not, due to time restrains.

A Service Template was created for the proxy service. This template included in Appendix A.1 contains all the information on the service attributes which the proxy service should advertise. Most of the attributes contained in the template are optional because they are not required for service usage. The only required attribute is `available-proxy-services` which is a multi-value string list of the requests which it proxies. This is not required for service usage, but was included since it will save unnecessary communications between the user agent and SA to determine if it will forward requests for specific services. In order to allow for best service selection, the algorithm requires that several attributes relating to the quality of the service agent be provided. This resulted in the creation of a concrete service template for optimal service selection, included in Appendix A.2. The selection of attributes for this template is discussed in Section 4.3 during the explanation of the best service selection algorithm.

An important end note to this solution is that the attribute data when registered with the DA, is static. It is only updated on service re-registration, implying that this approach limits the data "freshness" to the registration lifetime. A registration lifetime of 1 or 5 minutes should be selected because many of the attributes in squid are calculated once every 1 or 5 minutes. A lifetime of 1 minute was chosen to ensure that the directory agent has a "fresh data".

4.3 Directory Agent Modifications

Modifications to the directory agent were already completed when the service recommendation operations were added (Section 2.3).

4.4 User Agent Modifications

The UA selected was Mozilla [moz, 2000] because it provided the source code, making implementation possible. The first extension to the program was the creation of an option, allowing users to activate the dynamic proxy selection. This option window will also contain a default equation as to how the optimal proxy server should be found. Users have the option of keeping that predicate, or entering their own. Once the changes have been activated, a routine is activated. This routine frequently creates service requests for a proxy service with the predicate as entered by the user. Upon reception of a service registration from the DA of a single service, that service is activated as the new proxy server.

Due to time constraints, the above was not completely implemented. The issue preventing a full implementation was the activation of the proxy server within Mozilla. The reason for this is the enormous size of Mozilla, containing over 9500 different source code files, the routine to set the proxy server could not be found. Several queries to the Mozilla development team have been unsuccessfully. Once this issue has been resolved, it should work.

Chapter 5

Service Recommendation Performance Evaluation

This contains an evaluation of the modifications. It explains the simulation environment and how it was created to measure its performance.

5.1 Simulation Environment

It was not possible to test this algorithm within the environment which it would be used because it would have required hundreds of users with a modified web browser and several proxy servers. In order to show the performance of this algorithm, a simulation environment was created. This simulation environment consisted of a program simulating a proxy server and one simulating a web server as explained below. The simulation was performed by creating several proxies and browsers to model the real world as closely as possible.

5.1.1 Browser Simulator

The browser can not model a person's web browsing habits exactly because every browsing experience is different. In order to compensate for this, the times between web requests were generated from a random number in a range specified to the simulation. During execution of the browser, it can discovery proxy servers by static or dynamic means.

5.1.2 Proxy Simulator

The simulation of the proxy server acts very much like a proxy server does. It listens for requests from web browsers, and spawns off download tasks. Since, during a simulation, files can not be downloaded, a random number is generated within a range specified of how large the file is. Each download process waits until there is available bandwidth[1] and transfers part of the file then. This is repeated until the whole file has been transfered. Once the file transfer is complete, the proxy server logs how much it transfered, how long it took, and for whom it was transfered for.

5.2 Performance

The performance data as gathered by the simulation environment was used to determine the performance of the algorithm. The algorithm has been tested in several different cases. Each test shows how the results of the service recommendation, referred to as dynamic proxy selection, is compared to those of the static proxy server selection. For each test, the average delay times are contrasted by comparing their means and extraneous points (download tasks which took significantly more than the average time) by analysis of time versus delay time graphs.

5.2.1 Simulation 1

The first test illustrates the overhead this algorithm yields. This is done by observing how the algorithm compares against the the perfect balance (each proxy server with the exact same number of connections who are issuing roughly the same size of requests). This shows that the dynamic selection cost an average 2.29 seconds more than static selection. Furthermore, there is a large variation in the amount of time required to satisfy requests when dynamic configuration is deployed, as illustrated in Figure B.1, whereas

[1]Each download process and connected browsing program is competing for this limited bandwidth resource, just as in real proxy servers.

the static requests are all satisfied in about the same amount as shown in Figure B.2.

5.2.2 Simulation 2

The second test, whose parameters are contained in Table B.2, illustrates the performance of the algorithm against static proxy selection, where the load is unbalanced. The browsers with a static configuration, where 50% of the browsers are connected to one proxy server, and the remaining 50% are connected to two others, had an average delay of 293.40 seconds. The dynamic selection yielded an average delay of 22.40 seconds.

5.2.3 Simulation 3

This test attempts to determine how the algorithm performs against a distribution of proxy servers which are close to random. In this case, two proxy servers are started. For the static configuration, 55% of the clients are attached to one server, and 45% to the other. The dynamic selection saved an average of 39.82 seconds for each request for the conditions of the simulation given in Table B.4. Upon examination of the delay times as show in Figure B.5 and Figure B.6, there are a significant number of requests which had a delay of over 100 seconds for the static selection, whereas the dynamic algorithm handled all requests in under 100 seconds.

5.2.4 Simulation 4

The fourth simulation reflects a case where there is a large unbalance of the load among the static proxy servers. In this case, there were 150 browsers launched, of which 50 clients were attached to a single proxy, and four sets of 25 clients were attached to different proxies. The dynamic selection had an average delay of 1.88 seconds, while the static one had a delay of 3.50 seconds. The specifics are available in Table B.4.

Chapter 6

Conclusion

The creation of service recommendation facilities within SLP was a success. With the creation of four simple operators, it is now possible to query for the best service.

These new families were utilized through the extensions of a proxy server and a web browser. When these extensions perform together, web browsers automatically configure themselves to use of an optimal proxy server. During the lifetime of the browser, the proxy server will be changed invisibly to the user if a different proxy server can offer better performance.

An interesting finding while performing the simulations was the unexpected load balancing. It was originally thought that having UA requesting an optimal service from SAs would cause the load among the SAs to be uneven because the optimal service would become saturated. This assumption was completely wrong because as the UA began to consume the resources of the SA some UAs were migrated to other machines with available resources.

The evaluation of the extensions made, contained in Section 5, illustrated that this algorithm has a cost. When the dynamic proxy selection was compared to the optimal selection, it was shown to add delay times to transfers. As extensions were compared with other varying distributions, it was found that the extension out-performed unbalanced distribution of proxy servers to web browsers. In a real world environment, an optimal distribution of proxy servers to web browsers is very unlikely implying that the extensions will offer a performance increase for the client browsers.

6.1 Open and Future Issues

During the course of this project, several issues have been identified as open or requiring further work. This section includes the ones which are considered important by the author.

All of the approaches discussed in Section 3.2, 3.3, and 3.4 should be implemented. The implementation of all these approaches could determined if the placement of the selection algorithm within the DA was the optimal approach.

The function to select the optimal service runs in $O(n^2 \cdot m)$ time, where n is the number of proxy servers, and m is the amount of attributes contained by the proxy servers. An optimization of this routine would be to use proper searching routines.

The current method of using the rank of an attribute works well, but is not optimal since the rank does not tell you exactly how far one value is from another.

Another issue of future work could be to study which predicate for proxy servers returns the best service.

Appendix A

Templates

This appendix contains the templates that were created during this project.

A.1 Proxy Service Template

This is a template describes the attributes which a proxy server should advertise.

```
# Name of submitter: "David McCormack" <david.mccormack@ottawa.com>
# Date of creation: Fri Aug  4 15:54:10 EDT 2000
#
#   When used with SLPv2, this 'service:proxy:' template inherits all
#   of the security issues described in section 17 'Security
#   Considerations' of "Service Location Protocol, Version 2" [RFC 2609].
#
#   By advertising the security methods for each supported printer URL
#   the printer may expose information useful to attackers.  SLPv2
#   security or another security method SHOULD be used to authenticate
#   any service advertisements.

#----------------------------------------------------------------------

#  This document describes the proxy concrete type. The proxy service
#  provides access to web proxy.

# Introduction
#
#  This template is for use with a proxy server and with a browsers,
#  allowing for the dynamic discovery and selection of proxy servers.
#
#  An example service URL is the following:
#    service:proxy://sarah.scs.carleton.ca:3124
#

#----------------------------------------------------------------------
```

```
template-type=proxy

template-version=1.0

template-description=
    This is an concrete service type for discovery of proxy servers

template-url-syntax=
  url-path= ; an URL for the configuration data, as defined in RFC
            ; 1738 as amended by RFC 1808.

available-proxy-services = STRING M
#  This is a listing of all the potential proxy services, which
#  typically include www and ftp.

average-one-minute-kb-in-per-sec = INTEGER O
-1
#  The average kilobytes/second transfered into the proxy server
#  in one minute.

average-one-minute-kb-out-per-sec = INTEGER O
-1
#  The average kilobytes/second transfered out of the proxy server
#  in one minute.

average-five-minute-kb-in-per-sec = INTEGER O
-1
#  The average kilobytes/second transfered into the proxy server
#  in five minutes.

average-five-minute-kb-out-per-sec = INTEGER O
-1
#  The average kilobytes/second transfered out of the proxy server
#  in five minutes.

average-one-hour-kb-in-per-sec = INTEGER O
-1
#  The average kilobytes/second transfered into the proxy server
#  in one hour.

average-one-hour-kb-out-per-sec = INTEGER O
-1
#  The average kilobytes/second transfered out of the proxy server
#  in one hour.
```

```
cpu-usage = INTEGER O
-1
#  This is the average CPU usage of the proxy server.  It is a value
#  in the range [0, 100], with 0 being no usage, and 100 full usage.

program-name = STRING O
unknown
#  A string containing the name of the proxy server program.

http-hits-per-second = INTEGER O
-1
#  The average hits that the proxy server receives per second.

http-request-per-second = INTEGER O
-1
#  The average request per second that the server receives.

number-of-clients = INTEGER O
-1
#  The amount of currently connected clients to the proxy server.

available-band-width-in-kb-per-sec = INTEGER O
-1
#  The amount of the available bandwidth that is not currently in use
```

A.2 Optimal Proxy Service Template

This is a template describes the attributes which a proxy server should advertise so it can be used for an optimal server.

```
# Name of submitter: "David McCormack" <david.mccormack@ottawa.com>
# Date of creation: Fri Aug  4 15:54:10 EDT 2000
#
#   When used with SLPv2, this 'service:proxy:' template inherits all
#   of the security issues described in section 17 'Security
#   Considerations' of "Service Location Protocol, Version 2" [RFC 2609].
#
#   By advertising the security methods for each supported printer URL
#   the printer may expose information useful to attackers.  SLPv2
#   security or another security method SHOULD be used to authenticate
#   any service advertisements.

#----------------------------------------------------------------------


#  This document describes the proxy concrete type. The proxy service
#  provides access to web proxy.

# Introduction
#
#  This template is for use with a proxy server and with a browsers,
#  allowing for the dynamic discovery and selection of proxy servers.
#
#  An example service URL is the following:
#    service:proxy://sarah.scs.carleton.ca:3124
#

#----------------------------------------------------------------------


template-type=proxy

template-version=1.0

template-description=
    This is an concrete service type for discovery of proxy servers

template-url-syntax=
  url-path= ; an URL for the configuration data, as defined in RFC
            ; 1738 as amended by RFC 1808.

available-proxy-services = STRING M
#  This is a listing of all the potential proxy services, which
#  typically include www and ftp.

average-one-minute-kb-in-per-sec = INTEGER
-1
#  The average kilobytes/second transfered into the proxy server
#  in one minute.

average-one-minute-kb-out-per-sec = INTEGER
-1
#  The average kilobytes/second transfered out of the proxy server
#  in one minute.
```

```
average-five-minute-kb-in-per-sec = INTEGER
-1
#  The average kilobytes/second transfered into the proxy server
#  in five minutes.

average-five-minute-kb-out-per-sec = INTEGER
-1
#  The average kilobytes/second transfered out of the proxy server
#  in five minutes.

average-one-hour-kb-in-per-sec = INTEGER
-1
#  The average kilobytes/second transfered into the proxy server
#  in one hour.

average-one-hour-kb-out-per-sec = INTEGER
-1
#  The average kilobytes/second transfered out of the proxy server
#  in one hour.

cpu-usage = INTEGER O
-1
#  This is the average CPU usage of the proxy server.  It is a value
#  in the range [0, 100], with 0 being no usage, and 100 full usage.

program-name = STRING O
unknown
#  A string containing the name of the proxy server program.

http-hits-per-second = INTEGER
-1
#  The average hits that the proxy server receives per second.

http-request-per-second = INTEGER
-1
#  The average request per second that the server receives.

number-of-clients = INTEGER
-1
#  The amount of currently connected clients to the proxy server.

available-band-width-in-kb-per-sec = INTEGER
-1
#  The amount of the available bandwidth that is not currently in use
```

Appendix B

Data Results

The raw data is available with the programs as stored on the media. These tables below list the parameters for each of the tests and the significant results from each of the tests.

B.1 Simulation Results and Configuration

This section contains all of the parameters that were used for the simulations. Also included is a graph of the average delay times experienced by proxy servers in satisfying requests. The x axis denotes the time that the request was initially received from the browser, and the y axis displays the amount of delay experienced by that request. The legend of the graph contains the IP address of the proxy server. Each of the proxy servers have a different point representation on the graph. An optimum graph would be one where all the data points are close to the axis, which signifies very little delay experienced by the proxy server. Graphs containing a large range of y values are bad because this means that there is a large variation in the delay times for the request to be satisfied. There exists a section for each simulation containing a data table and a graph of the dynamic selection and then the static selection.

B.1.1 Simulation 1

This simulation relates the performance of the dynamic algorithm against a static algorithm where each proxy server contains an equal amount of browsers.

Proxy Servers	5
Client Browsers	100
Static Server Distribution	$[20\%, 20\%, 20\%, 20\%, 20\%]$
Dynamic Predicate	optimallow(ranklow(number-of-clients) +
	rankhigh(available-band-width-in-kb-per-sec))
Dynamic Re-Registration Time	30
Simulation Length	600
File Size	$[100, 150]$
Re-transmission Length	$[30, 60]$
Average Static Delay	1.36
Average Dynamic Delay	4.11

Table B.1: Simulation 1 Configuration and Results

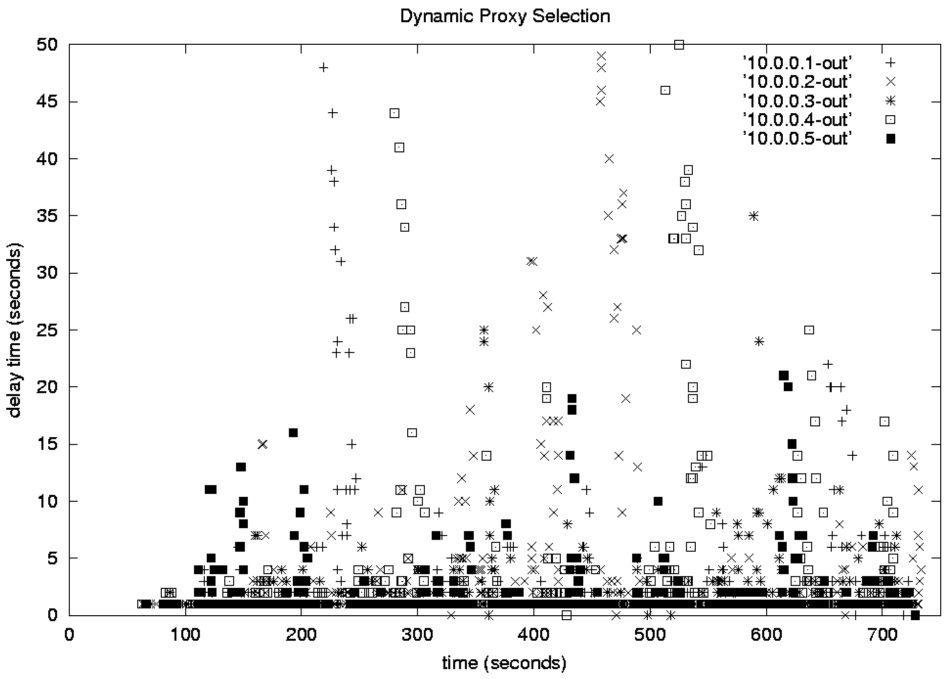

Figure B.1: Simulation 1 Dynamic Proxy Selection Delays for Browser Requests

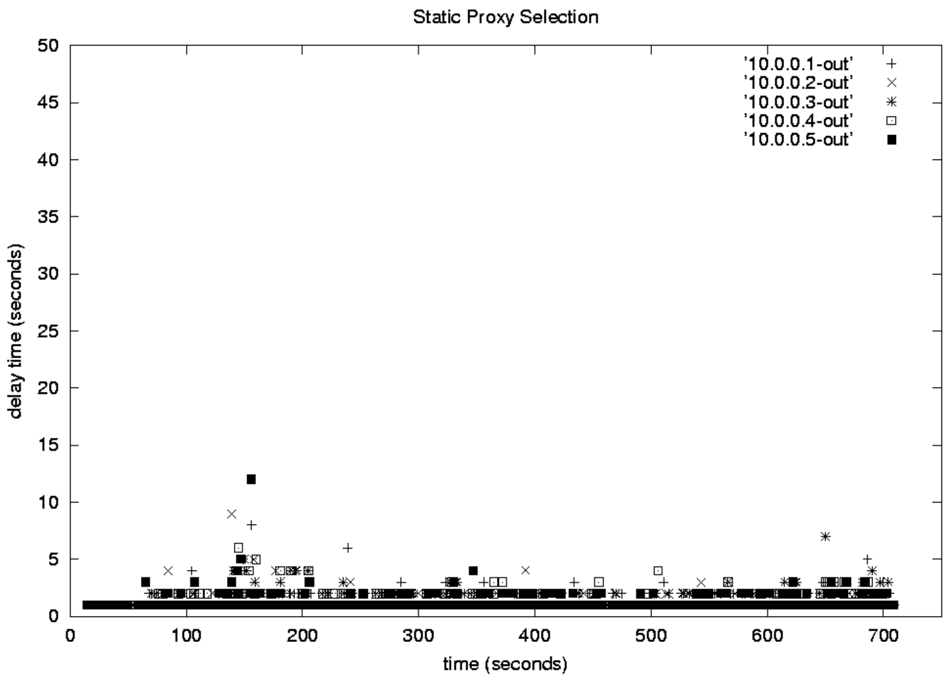

Figure B.2: Simulation 1 Static Proxy Selection Delays for Browser Requests

B.1.2 Simulation 2

The second simulation compares the performance of the dynamic selection algorithm against one where the static selection algorithm which is very unbalanced.

Proxy Servers	3
Client Browsers	100
Static Server Distribution	$[25\%, 25\%, 50\%]$
Dynamic Predicate	optimallow(ranklow(number-of-clients) + rankhigh(available-band-width-in-kb-per-sec))
Dynamic Re-Registration Time	40
Simulation Length	600
File Size	$[125, 175]$
Re-transmission Length	$[20, 40]$
Average Static Delay	293.43
Average Dynamic Delay	22.40

Table B.2: Simulation 2 Configuration and Results

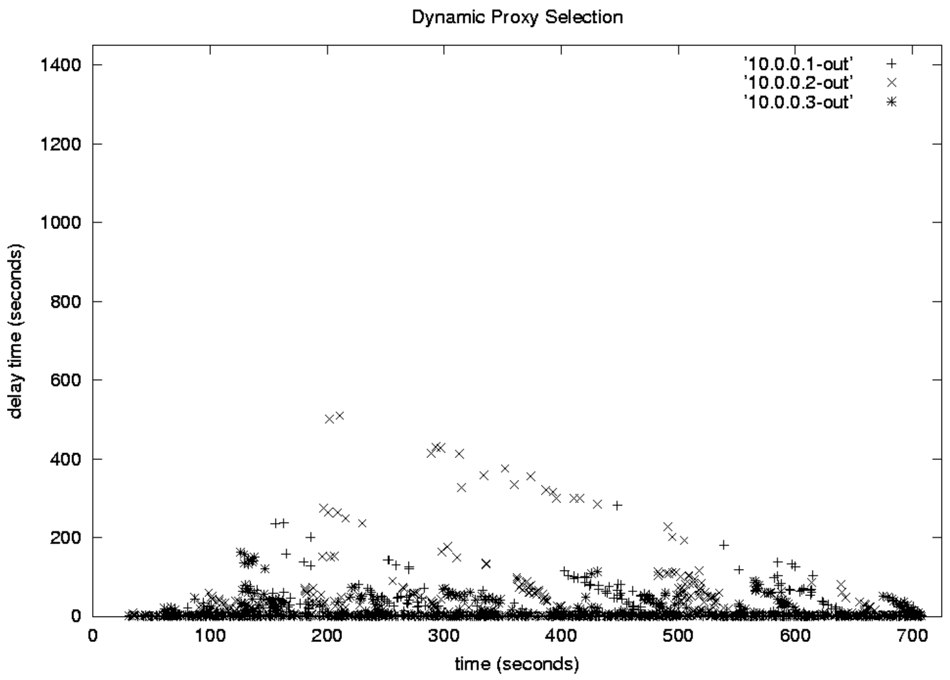

Figure B.3: Simulation 2 Dynamic Proxy Selection Delays for Browser Requests

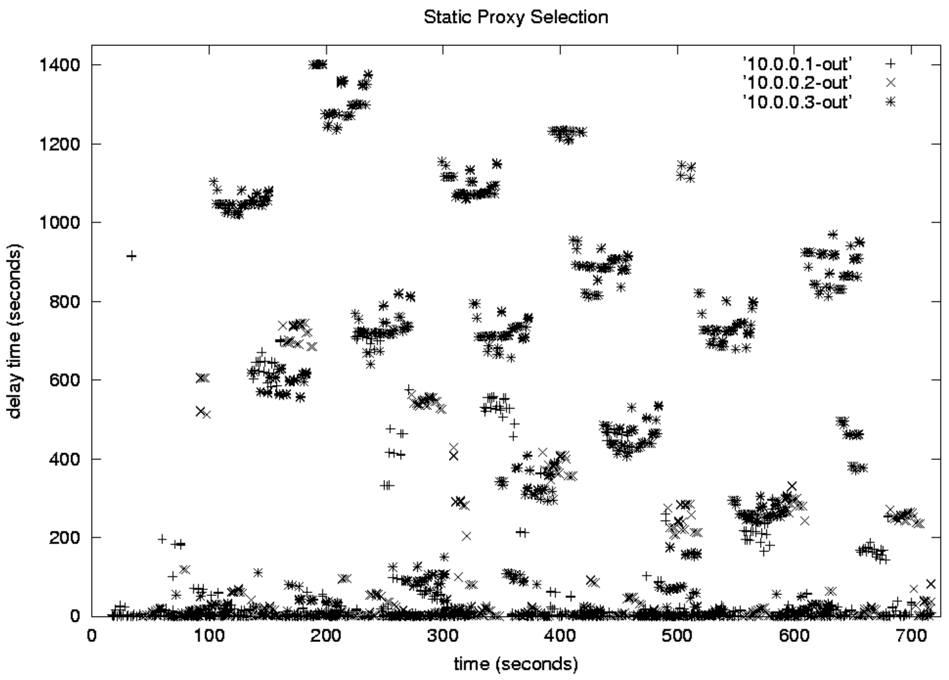

Figure B.4: Simulation 2 Static Proxy Selection Delays for Browser Requests

B.1.3 Simulation 3

This simulation contrasts the dynamic selection against the static selection when the static selection is fairly balanced.

Proxy Servers	2
Client Browsers	100
Static Server Distribution	$[45\%, 55\%]$
Dynamic Predicate	optimallow(ranklow(number-of-clients) + rankhigh(available-band-width-in-kb-per-sec))
Dynamic Re-Registration Time	1
Simulation Length	600
File Size	$[75, 100]$
Re-transmission Length	$[40, 60]$
Average Static Delay	44.92
Average Dynamic Delay	5.10

Table B.3: Simulation 3 Configuration and Results

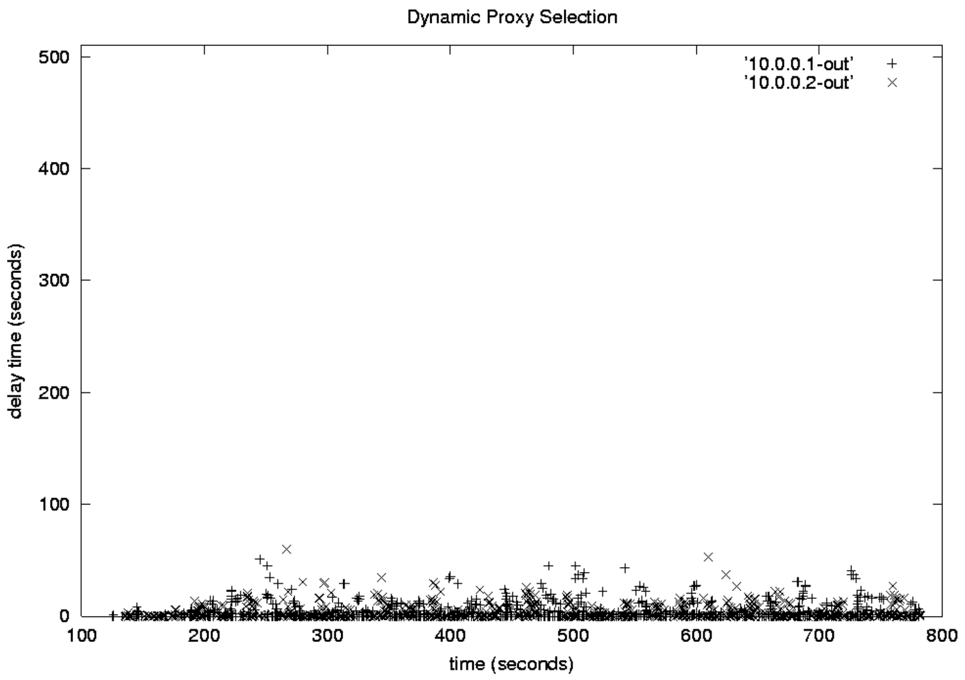

Figure B.5: Simulation 3 Dynamic Proxy Selection Delays for Browser Requests

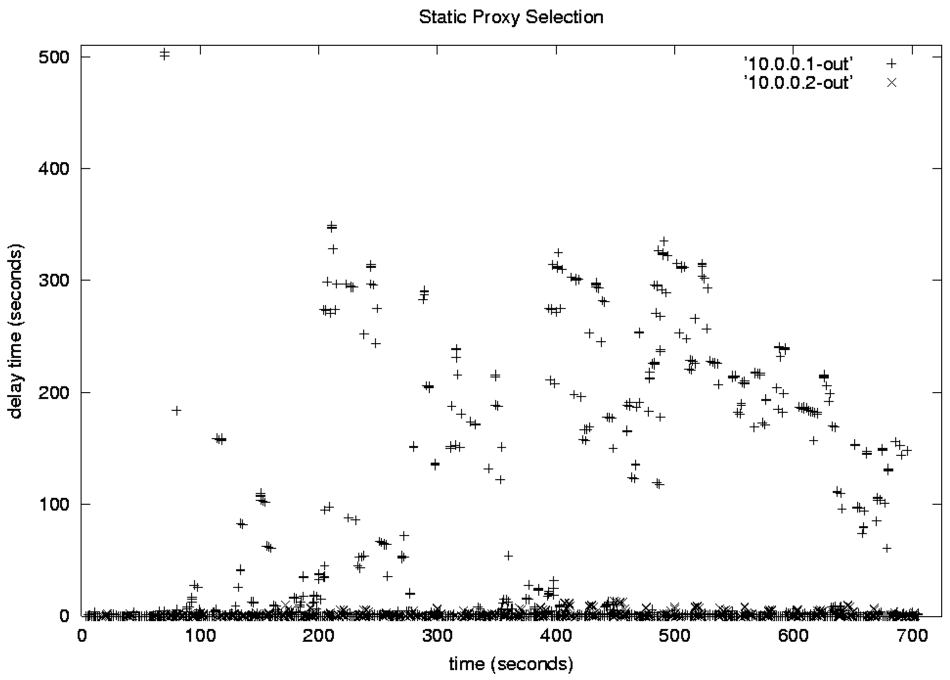

Figure B.6: Simulation 3 Static Proxy Selection Delays for Browser Requests

B.1.4 Simulation 4

The final simulation displays the differences in performance between the static algorithm with a moderately uneven distribution of proxy servers against the dynamic algorithm.

Proxy Servers	5
Client Browsers	100
Static Server Distribution	$[17\%, 17\%, 17\%, 17\%, 32\%]$
Dynamic Predicate	optimallow(ranklow(number-of-clients) +
	rankhigh(available-band-width-in-kb-per-sec))
Dynamic Re-Registration Time	1
Simulation Length	700
File Size	$[100, 125]$
Re-transmission Length	$[40, 60]$
Average Static Delay	3.50
Average Dynamic Delay	1.88

Table B.4: Simulation 4 Configuration and Results

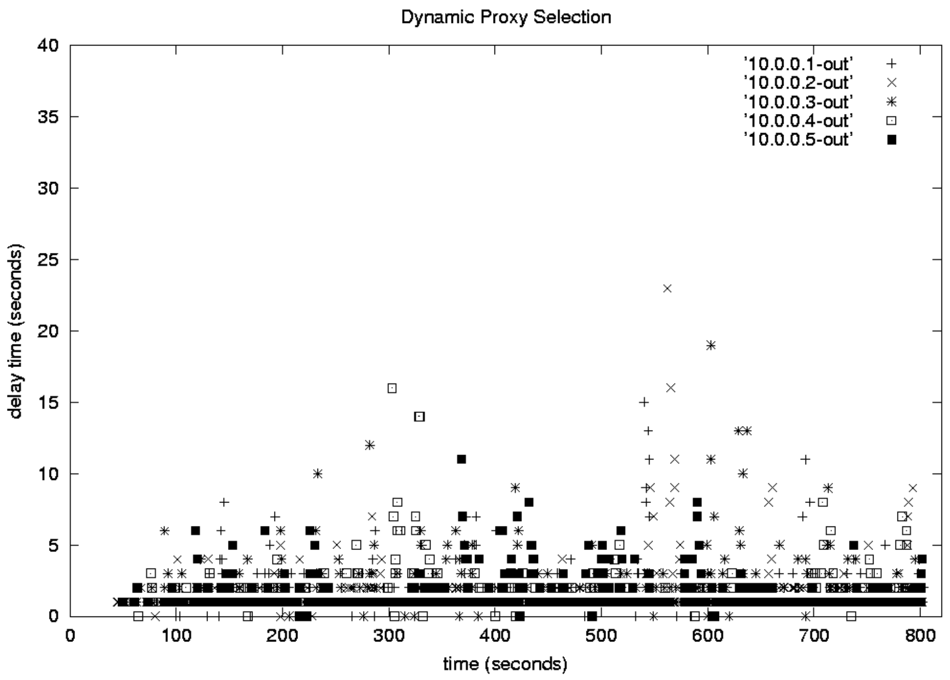

Figure B.7: Simulation 4 Dynamic Proxy Selection Delays for Browser Requests

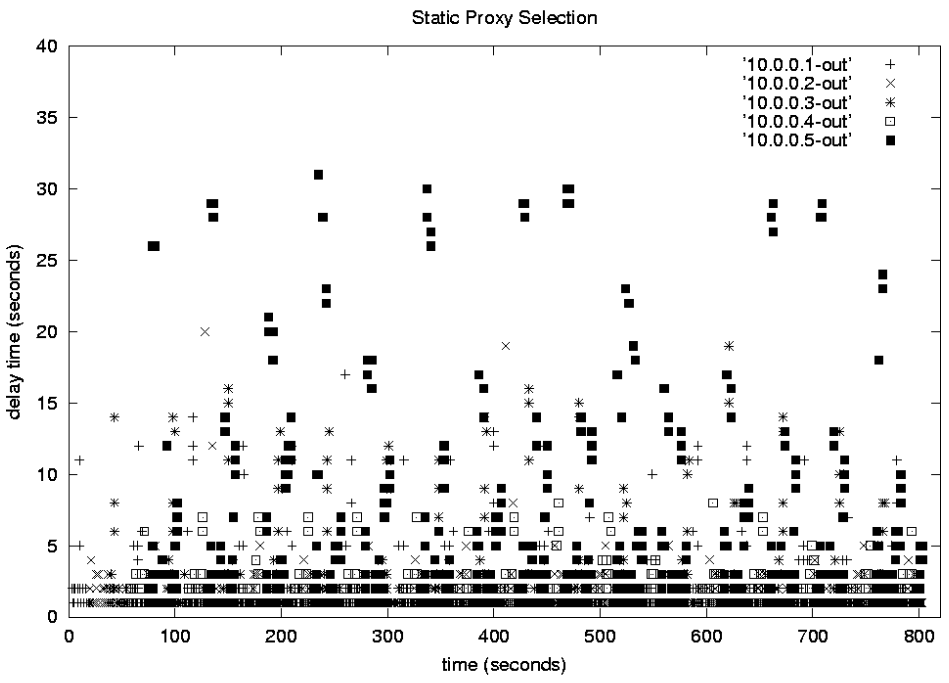

Figure B.8: Simulation 4 Static Proxy Selection Delays for Browser Requests

Bibliography

[Jin, 1998] (1998). Jini technology core platform specification. Technical report, Sun Microsystems, Palo Alto, United States of America.

[msi, 2000] (2000). Internet explorer home page. http://www.microsoft.com/windows/ie/.

[msp, 2000] (2000). Microsoft proxy server. http://www.microsoft.com/proxy.

[moz, 2000] (2000). mozilla.org. http://www.mozilla.org.

[net, 2000] (2000). Netscape products. http://home.netscape.com/download/index.html?cp=ncs.

[nsp, 2000] (2000). Netscape proxy server. http://home.netscape.com/proxy/v3.5/index.html.

[nov, 2000] (2000). Novell: Bordermanager enterprise edition. http://www.novell.com/products/bordermanager/.

[Tem, 2000] (2000). Service location template repository. ftp://ftp.isi.edu/in-notes/iana/assignments/svrloc-templates/. Central service location protocol repository repository.

[squ, 2000] (2000). Squid web proxy cache. http://www.squid-cache.org/.

[Guttman et al., 1999a] Guttman, E., Perkins, C., and Kempf, J. (1999a). RFC 2609: Serivce templates and service: Schemes. Status: Proposed Standard.

[Guttman et al., 1999b] Guttman, E., Perkins, C., Veizades, J., and Day, M. (1999b). RFC 2608: Serivce location protocol. Status: Proposed Standard.

[Howes, 1997] Howes, T. (1997). RFC 2254: The string representation of ldap search filters. Status: Proposed Standard.

[Hughes, 2000] Hughes, E. (2000). Service location protocol service browser. http://ssb.sourceforge.net/.

[Kempf and Goldschmidt, 2000] Kempf, J. and Goldschmidt, J. (2000). Notification and subscription for slp. Status: Draft.

[Wahl et al., 1997] Wahl, M., Howes, T., and Kille, S. (1997). RFC 2251: Lightweight Directory Access Protocol (v3). Status: Proposed Standard.

[Zhao and Schulzrinne, 2000] Zhao, W. and Schulzrinne, H. (2000). Interaction of slp directory agents for reliability and scalability. Status: Draft.