Local URL Resolution Protocol

Joseph Clark Ekstrom

Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Computer Sciences Commons

BYU ScholarsArchive Citation

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
LOCAL URL RESOLUTION PROTOCOL

by

Joseph C. Ekstrom

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
Brigham Young University
August 2006
BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Joseph C. Ekstrom

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date

Mark Clement, Chair

Date

Quinn Snell

Date

Dan Ventura
As chair of the candidate’s graduate committee, I have read the thesis of Joseph C. Ekstrom in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Mark Clement
Chair, Graduate Committee

Accepted for the Department

Parris Egbert
Graduate Coordinator

Accepted for the College

Thomas W. Sederberg
Associate Dean
ABSTRACT

LOCAL URL RESOLUTION PROTOCOL

Joseph C. Ekstrom

Department of Computer Science

Master of Science

DOGMA is a resource management system designed to create a supercomputer like system from unused desktop computers. Scalability is one of the challenges faced by DOGMA because it uses a strict client/server architecture. Distributing large files over a client server architecture is problematic since available network bandwidth is limited. The Local URL Resolution Protocol (LURP) addresses this problem for environments where there are high node densities. LURP implements a locality aware Peer-to-Peer file distribution model to increase the speed of file distribution while reducing the overall network congestion.
ACKNOWLEDGMENTS

I would like to thank my adviser Dr. Mark Clement and Dr. Quinn Snell for all their help and patience. I would especially like to thank my father, who has been an adviser, cheerleader, and a thorn in my side to finally get this finished.
Contents

Acknowledgments vi
List of Tables ix
List of Figures xi

1 Introduction 1
   1.1 DOGMA ................................................. 1
   1.2 Problem ............................................. 2
   1.3 Improvements ................................. 2
   1.4 Thesis Statement .............................. 4

2 Related Work 7
   2.1 Internet Protocols .......................... 8
   2.2 LAN File Systems ......................... 8
   2.3 WAN and Decentralized File Systems ......... 9
   2.4 Peer-to-Peer File Sharing .................. 9
   2.5 Multicasting ................................. 10

3 Local URL Resolution Protocol 11
   3.1 Local URL Resolution Protocol .......................... 12
      3.1.1 LURP Request Message ......................... 12
      3.1.2 LURP Response Messages ...................... 13
      3.1.3 LURP Message Flow ......................... 14
   3.2 LURP Implementation .......................... 18
List of Tables

1.1 Total Node Download Time Comparison ........................................ 3
List of Figures

3.1 A simple LURP example where the file is downloaded using LURP . 15
3.2 A simple LURP example where the file is downloaded using HTTP . 15
3.3 A LURP example of two nodes requesting the same file simultaneously 17
3.4 LurpDownloader inheritance hierarchy . . . . . . . . . . . . . . . . . . 18
4.1 Test Network Topology . . . . . . . . . . . . . . . . . . . . . . . . . . 22
4.2 Average bytes/microsecond for 15 Clients across 1, 10, and 100 Mbps . 24
4.3 Average bytes/microsecond for 100 Mbps for 15 and 30 Clients each using LURP and HTTP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
4.4 Comparison of runs consisting of 15 Clients at 1, 10, and 100 Mbps . 26
4.5 Comparison of 15 and 30 Clients running without a rate limiter . . . 27
4.6 Runs of 15 Clients on 1 Mbps . . . . . . . . . . . . . . . . . . . . . . . . 29
4.7 Runs of 15 Clients on 10 Mbps . . . . . . . . . . . . . . . . . . . . . . . 30
4.8 Runs of 15 Clients on 100 Mbps . . . . . . . . . . . . . . . . . . . . . . 31
4.9 Runs of 30 Clients on 100 Mbps . . . . . . . . . . . . . . . . . . . . . . 32
Chapter 1

Introduction

1.1 DOGMA

The Distributed Object Group Meta-Computing Architecture (DOGMA) harnesses the computational power of idle desktop computers, turning them into a distributed dynamic supercomputer. The two main components of DOGMA are the server and the clients. Currently the DOGMA Screen-saver Client is deployed on more than 600 Open-Lab computers across Brigham Young University campus.

DOGMA clients are normal desktop workstations that have the DOGMA Screen-saver installed. When users log off or are idle for a period of time, the DOGMA Screen-saver starts, allowing computational research tasks to be launched on these idle workstations. When a user logs on or becomes active, the DOGMA Screen-saver turns off, terminating all DOGMA launched tasks, giving complete control of the computer back to the user. Because DOGMA clients enter and leave when the screen-saver starts and ends, clients are continuously entering and leaving the system.

The DOGMA server is a resource manager designed to handle non-dedicated distributed computers. Its primary function is to allow for programs to be launched on idle desktop computers, allowing previously unused computers to be used as a computational resource. DOGMA has been built on top of open standards. The Hypertext Transfer Protocol (HTTP) is used to transfer files and data between the server and the client, and the Simple Object Access Protocol (SOAP) over HTTP is used for control message passing between the client and the server.

Clients connect to the DOGMA server in a polling fashion, asking for work.
Therefore, job launches in DOGMA are passive. Clients request work from the server, and the server assigns them tasks. DOGMA is not used for jobs that require extensive inter-node communication during the execution.

DOGMA clients only share a network connection. They do not share a filesystem or authentication domain. In order for a job to launch on a client, all the necessary executables and data files for the application must first be copied to the client.

1.2 Problem

DOGMA has proven its usefulness to the University and to the scientific community; however, it still has significant unrealized potential. Because DOGMA uses a strict client/server architecture, its scalability and efficiency is limited by the amount of available network bandwidth and the processing power of the DOGMA Server. These issues became apparent when DOGMA brought down the Computer Science firewall when launching a job across 600 machines. In order to increase the scalability of DOGMA, it is necessary to decrease both the number of connections to the DOGMA server and the amount of data transferred between the server and its clients. While it is difficult to lower the number and size of control messages between the server and the clients, improvements can be made in the efficiency of distributing files, such as code updates, executables, and computational task data. Improvements made to the file distribution mechanism, will not only lower the load on the server, but also minimize WAN usage, lower average download times, and consequently increase the amount of time each client can spend in computation.

1.3 Improvements

Because DOGMA uses standard network protocols, one solution to the file distribution problem is to install caching proxy servers on each LAN, decreasing WAN utilization and avoiding bottlenecks in the network. The following equations, while over-simplified, give an approximation for expected download times due to the
introduction of a file proxy server.

\[
time_{\text{default}} = \frac{p \times m}{b_l} \quad (1.1)
\]

\[
time_{\text{proxy}} = \frac{m}{b_l} + \frac{(p - 1) \times m}{b_h} \quad (1.2)
\]

In these equations, \(p\) is the number of clients, \(m\) is the message size, and \(b_H\) and \(b_L\) are \textit{high} and \textit{low} bandwidth link capacities. The high bandwidth link connects the clients of laboratory together. The low bandwidth link connects the server to laboratory networks. It is important to note that these equations make the following assumptions:

1. They deal only with the time to transmit the raw data over the wire, and do not take into account information such as time spent in routers and switches or in network protocol stacks.

2. File downloads happen at independent times, meaning clients do not compete for bandwidth of the link. If these downloads happened in unison, the time for each download would increase significantly, and the benefits of a proxy would be even more pronounced.

<table>
<thead>
<tr>
<th>(p)</th>
<th>(time_{\text{default}}) (b_h = 10\text{Mbps})</th>
<th>(time_{\text{proxy}}) (b_h = 10\text{Mbps})</th>
<th>(time_{\text{proxy}}) (b_h = 100\text{Mbps})</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20.97</td>
<td>5.87</td>
<td>4.36</td>
</tr>
<tr>
<td>10</td>
<td>41.94</td>
<td>7.97</td>
<td>4.57</td>
</tr>
<tr>
<td>20</td>
<td>83.89</td>
<td>12.16</td>
<td>4.99</td>
</tr>
<tr>
<td>50</td>
<td>209.71</td>
<td>24.75</td>
<td>6.25</td>
</tr>
<tr>
<td>100</td>
<td>419.43</td>
<td>45.72</td>
<td>8.35</td>
</tr>
</tbody>
</table>

\(b_l = 1\text{Mbps}\) and \(m = 512\text{KB}\)

Table 1.1 illustrates the potential benefit of installing a file proxy server. In this example, \(b_l\) is set at 1 Mbps, the size of the file \(m\) is set to 512 KB, a modestly sized file, while \(p\) and \(b_h\) are varied. Note that as \(p\) increases, \(time_{\text{default}}\) increases in
a linear fashion, however, when $b_h$ is either 10Mbps or 100Mbps the amount of time spent in downloading is much less than a linear increase.

While a file caching proxy would serve to decrease both server and network load, it is not a viable solution for DOGMA, or for GRID computing in general, because compute nodes participate in both logically and physically independent network domains. These network domains may be in different administrative domains and in geographically remote locations, making proxy server installation and administration impractical. Another solution is Peer-to-Peer file-sharing.

Bittorrent [4] is an example of how Peer-to-Peer computing can be implemented. Bittorrent facilitates the transfer of millions of files across the Internet by allowing users to download files from each other, decreasing the load on centralized servers. Bittorrent requires a centralized server for clients to connect to when they share information and files with peers. The clients then transfer files between themselves without directly involving the centralized server, which enables the central server to support a much larger number of clients. This is not the only benefit. Bittorrent has made improvements on previous Peer-to-Peer networks by allowing the client to download pieces of the file from multiple peers, and thus enable files to be transferred much more rapidly between the clients.

While Bittorrent is a good example of the power of Peer-to-Peer computing for transferring files, it still has a dependency on a centralized server for publishing file locations and seeding peer discovery. While many situations in Peer-to-Peer computing dictate that a centralized server be used, the deployment of DOGMA clients in a computer lab environment allow assumptions to be made that eliminate the need for a central server.

1.4 Thesis Statement

This research demonstrates that the Local URL Resolution Protocol reduces bandwidth, bypasses bottlenecks and allows DOGMA to scale more effectively. LURP combines the power of a local proxy server with the flexibility of Peer-to-Peer computing through using a proxy election protocol to organize peers without a central
server. The demonstration consists of testing an implementation of the protocol in a controlled network and comparing its performance with an existing implementation.
Chapter 2

Related Work

Extensive research has been performed to develop systems that efficiently distribute data over networks. It is possible to classify these distribution protocols into the following categories:

- Internet protocols - These protocols are designed for text/binary transfer across file system types. They depend on both a smart client and a smart server that perform various mapping functions between file operations on different systems.

- LAN file systems - These systems are typically designed for high performance under the assumptions that the network is very fast and has a very low error rate, conditions that are normally present on a building or a campus LAN. The basic design assumption is that the server is available and that communication between the client and server is very inexpensive.

- WAN file systems - These systems are typically designed for operation on slower networks with high latency and a higher probability of errors. The design often includes caching and update conflict resolution components so that operations can be performed in spite of network issues.

- Peer-to-Peer - These systems are typically designed to provide distributed access to content using resources provided by the clients of the service. The primary function of the system is to provide a directory or search space for locating content by name and then providing a direct download of content from the identified “peer” in the network.
• Multicasting - These systems are generally created for simultaneous transmission of content to multiple clients. Clients register for a “service” or transmission of some stream and then network resources are allocated so that all registered clients receive the data being multicast.

2.1 Internet Protocols

Classic file transfer protocols for the Internet, HTTP and FTP provide access for multiple clients from one server. These protocols provide efficient file transfers from a central server to distributed clients. Network bandwidth to a central server and server utilization prevent these protocols from supporting large numbers of clients.

Web caching[5, 6] addresses many of these challenges. Standard caching proxy servers are used throughout Internet to speed up web downloads. If proxy servers were used with DOGMA, it would require that a dedicated proxy server would have to be configured on each local network. The clients would also need to be configured to use those proxy servers. For these reasons, caching proxy servers are not a good solution for DOGMA.

2.2 LAN File Systems

The Network File System (NFS) [7], Andrew File System (AFS) [8], Server Message Block (SMB) [9], NetWare Core Protocol (NCP) [10], and Apple Filing Protocol (AFP) [11] are the most well known networked file systems. They are all examples of the classic client-server network file system. Clients of the file system communicate directly with the server, and cache a minimal amount of data. They are also characterized with locking mechanisms, which help maintain consistency among the clients. Because LAN’s typically have high bandwidth switched networks, extra communication is used as a tradeoff to ensure consistency.

These file systems require dedicated servers, which can affect performance adversely. FarSite [12], a Peer-to-Peer LAN file system, addresses many of these issues. It uses Byzantine replication algorithms to add fault tolerance, while providing security mechanisms to ensure that data is not compromised even when stored on
untrusted peers. FarSite is not able to handle cases where one third of the clients become unavailable. Because clients enter and leave the DOGMA system frequently and in large numbers, FarSite in not a valid solution for DOGMA.

2.3 WAN and Decentralized File Systems

LBFS [13], LegionFS [14], xFS [15], BuddyCache [16], and JetFile [17] provide decentralized file system services. The Low-bandwidth File System (LBFS) uses compression and delta techniques to lower the amount of traffic transferred across the network. LegionFS is middleware that allows multiple untrusting domains to use a common name space and security system to make data available across the GRID. LegionFS also makes use of defined local caches to speed data access for remote clients. While both the LegionFS and LBFS provide good remote file systems, they are too complex for DOGMA, and require large amounts of administrative overhead. They also are limited in that they require servers or caches on local networks.

BuddyCache [16] uses a Peer-to-Peer type cache to reduce the number of requests made over a WAN. BuddyCache has a “redirecting” component which resides on one of the peers in the network or on a designated node, which redirects page requests to peer caches, or the main server. Because BuddyCache uses designated “redirectors”, it requires a significant amount of configuration.

JetFile [17] uses multicast to distribute files to smart file caches which are located on remote networks. High speed network smart caches receive data before files are requested, while caching policies are used on slower networks to reduce file access latency. A reliable multicast is used for client-to-client updates to reduce dependence on centralized servers. Like LBFS [13], LegionFS [14], and BuddyCache [16], JetFile [17] uses proprietary file transport mechanisms, making them less desirable as components in other systems.

2.4 Peer-to-Peer File Sharing

Although Peer-to-Peer file sharing was first popularized by Napster [18], other protocols, like Gnutella and Bittorrent [4] are more popular today. Napster required
a central server, which managed the peers and provided file searching capabilities. Gnutella took the next step and removed the central server. Instead, it requires a well known master nodes to connect new peers. Searches are performed by sending queries through the peer mesh. Bittorrent requires a “tracker”, which manages all the peers downloading a particular file; however, searching for data is not intrinsic to the protocol. The biggest contribution of Bittorrent is that it allows different parts of the same file to be downloaded simultaneously from different peers.

Squirrel [19] is a Peer-to-Peer web caching solution which is based on Pastry [20], a self organizing Peer-to-Peer network. It eliminates the need for configured web proxies on the LAN; however, it requires each client to join a global Peer-to-Peer group through a central server. This central server allows clients to seed peers and discover available services.

None of these protocols deal sufficiently with the problem of client locality and all of them require human interaction and configuration. These characteristics are unacceptable for DOGMA.

2.5 Multicasting

Multicasting [21] is very similar to broadcasting UDP traffic. It adds reliability and the ability to span broadcast domains. A single packet from a sender is received by multiple recipients. Multicast recipients must “join” a multicast group before they will receive any data. While multicasting is very useful for sending large amounts of data to multiple recipients, it has a few drawbacks. All routers between the sender and receiver must be configured to forward multicast traffic. Another drawback is that the receivers must be “listening” when the message is being sent. It is not an on demand protocol, like most file transfer protocols. Because DOGMA runs on independently managed networks, and nodes join and leave frequently, multicast is not a viable solution.
Chapter 3

Local URL Resolution Protocol

LURP has been designed specifically for use with DOGMA as deployed across the Brigham Young University campus. It can also be used at other large commercial or educational institutions. The DOGMA Client is simple, lightweight, and requires little setup or configuration. At BYU, DOGMA clients become available when they are not being used by students or faculty members, as indicated by the DOGMA screen-saver turning on. Most of these clients are computers located in open computer laboratories. In these labs, machines are connected to each other by a high speed switch, and participate in the same Ethernet and hence User Datagram Protocol (UDP)[21] broadcast domain. Data access to the rest of campus has a lower bandwidth connection, making communication between machines in the same lab faster than communications with servers outside the Local Area Network (LAN). For the purposes of this section, a DOGMA client is synonymous with a LURP node. The DOGMA Client will be running a LURP Node which consists of both client and server pieces.

LURP was patterned after the Address Resolution Protocol (ARP)[22][21]. ARP is used on Ethernet networks to translate Internet Protocol (IP) addresses to Ethernet addresses. This is accomplished by a node broadcasting an ARP Request to the Ethernet, then waiting for a unicast ARP Response from the node with the corresponding IP address. LURP has two types of messages that are sent over UDP between LURP nodes that are on the same subnet, namely LURP request and LURP Response messages.
3.1 Local URL Resolution Protocol

LURNP nodes communicate with each other reliably, using Transmission Control Protocol (TCP)\textsuperscript{[21]}, and unreliably using UDP. UDP is used to locate files by their URL, and to respond to location requests. TCP is used to transfer files between two LURNP nodes. A request message is sent to locate a file represented by a URL. This request is broadcast to all participating LURNP clients on the subnet. When those clients receive the request, they respond directly to the sending client, if they have the requested file. If a client does not have the requested file, it will not respond to the request message. The requesting client then waits for responses from neighboring clients. If the requesting client receives a response, it will then download the resource from a neighboring client, otherwise it retrieves the file from the original URL. Once the requesting client has downloaded the file, it is able to respond to other client’s requests for that file on the subnet.

3.1.1 LURNP Request Message

LURNP requests are sent to locate a file, uniquely identified by it’s URL, on the LURNP node’s subnet. LURNP Request messages have the following parts:

| File URL | MD5 |

1. File URL - The original URL for the file. This is used to uniquely identify the file.

2. File MD5 sum - This is an optional mechanism for file validation.

While the MD5 is optional, it provides a mechanism for peers to validate that the file it has cached is a current version. Because LURNP requests are only used to locate files, each will be referred to as LURNP\_LOCATE message. A LURNP\_LOCATE is a UDP broadcast to all participating nodes on the LAN to query for a specific file. A broadcast UDP message is used to send the request to all LURNP nodes simultaneously, and without having to “know” what nodes are participating.
3.1.2 LURP Response Messages

A LURP Response is a UDP message sent directly from one client to another if and only if the replying client has the file associated with the URL that was requested. LURP Response messages have the following parts:

<table>
<thead>
<tr>
<th>File URL</th>
<th>MD5</th>
<th>Location</th>
<th>State</th>
<th>Node Weight</th>
</tr>
</thead>
</table>

1. File URL - The original URL for the file. This is used to uniquely identify the file.

2. File MD5 sum - This is an optional mechanism for file validation.

3. File Location - The location of the file on the local subnet.

4. File State - The current state of the file on the client. It can be one of three values: REQUESTING, DOWNLOADING, or CACHED. The REQUESTING Node State and the Node Weight were added because jobs were launching on multiple clients on the same LAN within a short period of time. Without the REQUESTING and DOWNLOADING state, the clients would compete for bandwidth from the central server if none of the clients had the file in a CACHED state.

5. Node Weight - A random number generated to aid in peer elections for file downloads.

The file URL and MD5 are sent in the LURP response message so the requesting LURP node can verify information about the file prior to downloading it. The File Location contains information necessary for the requesting node to download the file from the responding node. The File State field contains information about the current state of the file on the client. Without the REQUESTING and DOWNLOADING state, the clients would compete for bandwidth from the central server if none of the clients had the file in a CACHED state.

The Node Weight field is a random number generated by the client for each file it downloads. It is used when all LURP Responses have a File Status of REQUESTING. The node weight is used by the clients to elect one of the REQUESTING clients to download the file from the central server. The node with the lowest Node Weight
wins the election and downloads the file from the central server, while the other peers
wait to download the file from the winner.

3.1.3 LURP Message Flow

When a DOGMA client is instructed to start a job which requires File A, it
then acquires the file using LURP, in the following manner:

1. Send a broadcast LURP_REQUEST.

2. Wait for LURP Responses.

3. Download the file either from a peer, or from the central server.

The following examples illustrate the LURP Protocol.

Simple LURP Examples

Figure 3.1 shows a simple example of a LURP message exchange, where the
end result is the file being downloaded using LURP from a peer node.

1. **Requester** broadcasts a LURP_LOCATE message, and the message is received
   by participating LURP nodes on the subnet.

2. **Peer 1** and **Peer 2** respond to the **Requester** with a LURP_CACHED message.

3. **Requester** downloads the file from **Peer 1**

   Figure 3.2 shows a simple example of a LURP message exchange where the
   request times out.

1. **Requester** broadcasts a LURP_LOCATE message to the UDP broadcast domain.

2. **Requester** waits for response messages, but eventually times out.

3. **Requester** downloads the file using the URL from the **HTTP Server**
Figure 3.1: A simple LURP example where the file is downloaded using LURP

Figure 3.2: A simple LURP example where the file is downloaded using HTTP
LURP Message Flow Details

First, the LURP client creates a UDP socket. When a UDP socket is created, a random number between 1000 and 65536 is assigned as the source port for packets sent from that socket. The client broadcasts a LURP_LOCATE message over the UDP socket to the specified LURP UDP port, which is currently 7677.

The LURP client then waits for a short configurable period to allow neighboring clients on the LAN to respond to the LURP Request message. The client listens for LURP Response messages on the UDP socket that the LURP Request used. In the reference implementation, the timeout period is set to 100 ms. If a CACHED_LURP Response is received, the client will download the file from the responding neighboring client. If a DOWNLOADING_LURP Response is received, the client will wait for another specified timeout, then start the process again by sending a broadcast LURP Request. If only REQUESTING_LURP Responses are received, then the LURP client will download the file from the central server if and only if the client’s Node Weight is smaller than all Node Weights received in the LURP Responses. Otherwise, REQUESTING_LURP Responses are processed like DOWNLOADING_LURP Responses. If no response is received, then the file is downloaded directly from the central server.

LURP Election Example

Figure 3.3 shows a complex example of a LURP message exchange where two nodes are simultaneously requesting the same file.

1. **Requester** and **Requesting Peer** broadcast LURP_LOCATE messages, and the message is received by participating LURP nodes on the subnet.

2. **Requester** and **Requesting Peer** each respond to the other client’s LURP_LOCATE message with a LURP_REQUESTING message.

3. **Requester** wins the election process by having a lower weight value.

4. **Requesting Peer** loses the election and begins to wait for the timeout interval.
Figure 3.3: A LURP example of two nodes requesting the same file simultaneously

5. **Requester** downloads the file from the **HTTP SERVER**

6. **Requesting Peer** re-broadcasts the LURP_LOCATE message.

7. **Requester** now has the file cached, and responds with a LURP_CACHED message to the **Requesting Peer**.

8. **Requesting Peer** then downloads the file from **Requester**

3.2 LURP Implementation

This section describes a Java implementation of LURP that is integrated into DOGMA. It was written in Java in order to be easily embedded into the DOGMA framework.

The Dogma Client, which runs on all Dogma Client nodes, was written in Java to be cross platform and was designed to be modular, extensible, and simple to
All referenced classes are sub-packages to the edu.byu.cs.ncl.dogm3.client package. In this package, there is a DogmaDownloader interface located in the interfaces subpackage. This interface is used by the Dogma3 Class to download the needed files. Because basic HTTP functionality was necessary, the BasicDownloader class was extended by the lurp.LurpDownloader class as shown in Figure 3.4. The BasicDownloader class implements the DogmaDownloader interface and is the default DogmaDownloader interface used by the DOGMA Client. It supports downloading files using the HTTP protocol.

The LurpDownloader was logically divided into three main parts: the LurpFileMap, the LurpClient, and the LurpServer. The LurpFileMap stores and manages files that the LurpClient downloads, and that the LurpServer sends. Each file that is downloaded by the LurpClient is stored at the value in the map as a LurpFile. The LurpFile class stores information about the file, such as URL, MD5, and the current status (DOWNLOADING,Cached,REQUESTING). The map is keyed with the URL of the file, so that the server can easily check for cached files and respond to other Clients LURP_REQUEST messages.

The LurpClient is responsible for downloading files, and has a pointer to the
LurpFileMap. When it receives a request to download a file, it creates a LurpFile, sets the LurpFile's state to REQUESTING, and adds it to the LurpFileMap. It then sends and waits for messages as described in Section 3.1.2. It then updates the status of the LURP_FILE to DOWNLOADING, then downloads the file through LURP or HTTP. Once that is finished, it updates the status of the LurpFile to CACHED.

The LurpServer responds to LURP_REQUEST messages and processes file download requests from the other clients. It has two threads, one listening for and responding to LURP_LOCATE broadcast UDP packets, and one is reliably serving the cached files over TCP connections.

The LurpMessage class represents a LURP message that was or will be transported over the network. It contains a LurpFile and the type of request. The LURP Request is of the form:

"LURP_REQUEST ${FILE URL};${FILE MD5 SUM}"

The LURP Response consists of the file’s status, the file’s download URL, the file’s MD5, and the file’s location as follows:

"LURP_RESPONSE ${file status};${file URL};${file MD5};${file location}-${node weight}"

The file location is formatted using the URL syntax in the following way:

lurp://node_address:port/filename
Chapter 4

LURP Performance

4.1 Test Environment

The environment used for assessing the validity of LURP, consisted of four parts: a DOGMA Server, an HTTP server, a bandwidth limiter, and DOGMA clients (Figure 4.1). The DOGMA Server was running JBoss version 3.2.5, against a Hypersonic database, with a version of Dogma3 built from the Subversion repository. The server machine was a Dual 2.4 Ghz Intel P4 Xeon, 1 Gig of RAM, and a 100 Mbps NIC.

The HTTP server was running Apache 2.0.54. The server was a 500 Mhz PIII, with 512 MB of RAM, and a 100 Mbps NIC. It was connected to the bandwidth limiter, not to the normal network, so a lower bandwidth link could be simulated.

The bandwidth limiter was a computer with 2 network cards. It was configured to forward and limit the bandwidth of all incoming and outgoing traffic to the HTTP server. Iptables\textsuperscript{[23]} and tc\textsuperscript{[24]} were used to set the incoming and outgoing bandwidth to values less than 100 Mbps, but reset to default values when testing 100 Mbps.

The DOGMA Client was run on Computer Science Department Linux lab machines. These Linux machines are standard desktop computers. During the test runs, these machines ran two different versions of the client, one LURP enabled, and the other the standard client.

4.2 Experiments

DOGMA jobs were created with the following parameters:
Figure 4.1: Test Network Topology
Figure 4.2: Average bytes/microsecond for 15 Clients across 1, 10, and 100 Mbps for LURP and HTTP

- The *filesize* of the data file to be downloaded. The file sizes used were 1, 2, 4, 8, 16, 32, 64, 128, 256, and 512 kilobytes, and 1, 2, 4, 8, 16, and 32 megabytes. File sizes are all powers of 2 bytes, starting at $2^{10}$ and ending at $2^{25}$.

- The *number of clients* each job should run on. 15 clients for 1, 10, and 100 Mbps, and 30 clients at 100 Mbps.

- The *type of DogmaDownloader* used to download files. The the downloaders were BasicDownloader(HTTP) and the LurpDownloader.

- The *bandwidth* to the HTTP server. 1, 10, and 100 Mbps.

Each job was then run 5 times.
Figure 4.3: Average bytes/microsecond for 100 Mbps for 15 and 30 Clients each using LURP and HTTP

### 4.3 Results

Figure 4.2 shows the average rate for file downloads with 15 clients across the 5 runs, with the HTTP server link set to 1, 10, and 100 Mbps. The 1 Mbps runs of LURP and HTTP show a significant increase in download speeds using LURP. It is also interesting to note that the 1 Mbps LURP run performs similarly to standard HTTP at 10 Mbps, or about 9 times faster than HTTP at 1 Mbps. In the 10 Mbps runs, LURP performs more than 3 times better than HTTP. However, in the case of 100 Mbps, or in other words, when there is no bandwidth limitation between the clients and the HTTP server, HTTP performed better overall.

In Figure 4.3, the download rates of LURP and HTTP are compared for 15 and 30 clients. Notice that as the number of clients increases, the average download rate decreases as the size of the file becomes larger than 8 MB. In both Figure 4.2 and 4.3, there is a noticeable spike in performance for medium sized files on LURP runs. For
the 1 Mbps runs, the spike falls between 128 KB and 512 KB, while for the 10 and 100 Mbps runs, it falls between 128 KB and 4 MB. When the entire file can be downloaded before the majority of the clients begin requesting the file, LURP performs much better, causing the spikes in performance. For the DOGMA environment, where clients request work in a quasi-random fashion, LURP performance should be very good, possibly better than these results show. Currently, HTTP file downloads must be completed before the file can download from a peer. If the clients streamed data to peers as it was received, all clients downloading the file would receive it at almost the same rate as one HTTP download. This would address the performance issues that cause LURP to perform slightly worse than an HTTP server on the LAN.

Figures 4.4 and 4.5 show the percentages of files that were downloaded using LURP or HTTP. As the file size increases to 32 MB, the number of files downloaded from a peer client goes down, while the number of files downloaded from the server using HTTP increases. This is very pronounced in Figure 4.5 for 30 clients. An
Figure 4.5: Comparison of 15 and 30 Clients running without a rate limiter

Figure 4.6: Runs of 15 Clients on 1 Mbps
unknown problem was encountered during testing with the java.nio package, which caused LURP downloads to terminate prematurely, causing the file to be downloaded using HTTP. This problem could explain why, as the size of the file increases, the chance of the LURP download failing increases, increasing the number of files downloaded using HTTP.

Figures 4.2 and 4.3 show a general trend that after the “spike”, LURP performance begins to taper off. There is a direct correlation between the number of files downloaded from peers using LURP and overall performance. The more files downloaded from peers, the better LURP performs.

Figures 4.6, 4.7, 4.8, and 4.9 show the five runs using both the LURP and standard HTTP enabled clients. It is interesting to note that behavior is much more erratic when running at 100 Mbps, or when the HTTP server is located on the LAN. In that case, LURP did not perform as well as the standard HTTP server. While the theoretical performance of LURP is better than HTTP, the current implementation
Figure 4.8: Runs of 15 Clients on 100 Mbps

Figure 4.9: Runs of 30 Clients on 100 Mbps
does not realize the full potential of the protocol. However, LURP with an HTTP server located over a slower link demonstrates the potential of the protocol.
Chapter 5

Contributions and Future Work

5.1 Contributions

This thesis develops the Local URL Resolution Protocol, and provides a Java implementation for use with the DOGMA Client. The Java implementation works well when the bandwidth of the network connecting the peers is greater than the bandwidth connecting the clients to the central server. LURP showed speed improvements of 9 times with the WAN bandwidth limited to 1 Mbps and over 3 times when limited to 10 Mbps. While there are still ways that LURP can be improved, LURP has been shown to increase the speed of file downloads and decrease network congestion.

5.2 Future Work

An optimization that would further decrease download times for LURP would be to “stream” the data without waiting for the HTTP download to finish. Streaming support would remove the necessity of completely downloading the file before beginning to send the file to other clients on the LAN. All clients downloading the file would receive it at almost the same rate as one HTTP download.

The number of simultaneous downloads that are initiated could be limited. This would speed up individual file downloads by reducing network bandwidth conflicts, but could cause other clients to wait longer to begin downloading. However, combined with the streaming optimization previously mentioned, the client wait to begin a download could be avoided as well.
Bibliography


