A Method for Efficient Transmission of XML Data across a Network

Alexander Ridgewell
School of Information Sciences and Engineering, University of Canberra, Masters Degree
Copyright

Copyright @ 2007 University of Canberra (UC), Australia

All rights reserved. Printed by the University of Canberra. This publication is protected by copyright, and permission must be obtained from the publisher prior to any reproduction.
For information regarding permission, write to:

University of Canberra
University Drive
Bruce ACT 2617
Australia
Abstract

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879), which is a well defined, public standard. It uses plain text to encode a hierarchical set of information using verbose tags to allow the XML document to be understood without any special reader. The use of schemas in XML also allows a well defined contract describing what a single XML document means. The self-contained nature of XML and the strong contract provided by its schemas makes it useful as an archival storage format and as a means of communicating across system or organizational boundaries. As such XML is being increasingly used by businesses throughout the world. These businesses use XML as a means of storing, transmitting and (with the use of style sheets) displaying information.

The simple, well defined structure of XML does present some problems when it is used by businesses and similar organizations. As it is an open, plain text based standard care must be taken when looking at security. The use of plain text with verbose tags also results in XML documents that are far larger than other means of storing the same information.

This thesis focuses on the affect of the large size of XML when it is used to communicate across a network. This large size can often increase the time taken to transmit the document and we were interested to see how it could be minimized. We investigated the ways that are used to control the size of XML documents and how they are transmitted.

We carefully investigated by implementing solutions on how to transmit the XML document. We then first presented a new method, called dynamic adaptive threshold transmission (DATT), in comparisons with other existing similar methods, which, under the discussed conditions, offers significant improvements in transmission times and network transmission efficiencies.
Certificate of Authorship of Thesis

Except where clearly acknowledged in footnotes, quotations and the bibliography, I certify that I am the sole author of the thesis submitted today entitled –

A Method for Efficient Transmission of XML Data across a Network

I further certify that to the best of my knowledge the thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

The material in the thesis has not been the basis of an award of any other degree or diploma except where due reference is made in the text of the thesis.


..........................................................................
Signature of Candidate
..........................................................................

..........................................................................
Signature of chair of the supervisory panel

Date: ..............................................................
Acknowledgements

I would like to acknowledge my great appreciation of my supervisors Dr Xu Huang and Dr Dharmendra Sharma of the University of Canberra for all their advice, aid and encouragement during the research and writing of my Masters thesis.

I would also like to express my appreciation of the Department of Education, Science and Training (DEST) as well as the University of Canberra for providing the Research Training Scheme Scholarship that allowed me to carry out my Masters work.

Finally, I would like to thank my family for the support they have given me over the course of this degree, and for proofreading of the final thesis.
Contents

Copyright ........................................................................................................................... 2
Abstract .......................................................................................................................... 3
Certificate of Authorship of Thesis ................................................................. 4
Acknowledgements ............................................................................................... 5
Contents ....................................................................................................................... 6
List of Figures ............................................................................................................... 8
List of Tables ............................................................................................................... 9
Glossary of Abbreviations .................................................................................... 10
Chapter 1 ..................................................................................................................... 11
Introduction .................................................................................................................. 11
  1.1 Motivation ............................................................................................................ 13
  1.2 Thesis Argument ............................................................................................... 14
  1.3 Limitations of the study .................................................................................... 14
Chapter 2 ..................................................................................................................... 16
About XML Encoding and Transmission ........................................................... 16
  2.1 Encodings used with XML .............................................................................. 16
  2.2 Security and Access control .......................................................................... 17
  2.3 Network Characteristics ................................................................................. 18
  2.4 Usage of XML in Business to Business processes ....................................... 19
Chapter 3 ..................................................................................................................... 21
Current Efficient Methods for XML Transmission ..................................... 21
  3.1 Element Name Replacement ......................................................................... 21
  3.2 Efficient XML Interchange .......................................................................... 22
  3.3 File Compression ............................................................................................ 24
  3.4 Network Adaptable Middleware (NAM) ....................................................... 25
Chapter 4 ..................................................................................................................... 27
Dynamic Adaptive Threshold Technique (DATT) Formatting ................... 27
  4.1 The Theory of DATT ....................................................................................... 28
       4.1.1 Using a Threshold Value to decide if Compression is needed. .......... 28
       4.1.2 A Dynamic Threshold ....................................................................... 31
  4.2 Testing DATT .................................................................................................... 33
       4.2.1 Determining the Gain from Compression .......................................... 34
4.2.2 Using a Simple Threshold Value to determine if Compression is needed –
   The One Pass Technique (OPT) ................................................................. 41
4.2.3 Extending Simple Threshold Value to the DATT ............................ 50

| Chapter 5 ......................................................................................................... 55 |
| Discussion ........................................................................................................ 55 |
| Chapter 6 ......................................................................................................... 57 |
| Conclusion and Further Work ........................................................................ 57 |
| Bibliography .................................................................................................... 60 |
| Appendices ....................................................................................................... 65 |
| Appendix A: My Contributions ..................................................................... 65 |
| Appendix B: Program Code ........................................................................... 66 |
| Appendix C: Sample Data .............................................................................. 78 |
List of Figures

Figure 1: Increase in XML usage...................................................................................... 12
Figure 2: Curve fitting example sets of uncompressed and compressed transmission data to determine the Threshold Value................................................................... 29
Figure 3. An empty SOAP XML document ................................................................. 35
Figure 4. A SOAP XML document with a single entry............................................... 36
Figure 5. A CSV file holding a single entry ................................................................. 37
Figure 6: Experimental network setup............................................................................. 44
Figure 7: Total NAM transmission time vs. total OPT transmission time. ...................... 46
Figure 8: NAM decision time vs. OPT decision time...................................................... 47
Figure 9: NAM total time vs. OPT total time................................................................. 48
Figure 10: Transfer times for a network with Threshold Value 425, using ±10% and ±20% of the Threshold Value for OPT decisions.................................................. 49
Figure 11: DATT decision time ...................................................................................... 52
Figure 12: NAM decision time. ..................................................................................... 53
List of Tables

Table 1. XML Binary Characterisation requirements [27] .................................................. 23
Table 2. Size of an empty SOAP message ........................................................................ 37
Table 3. Size of a SOAP message with one entry ............................................................ 38
Table 4. Size of a SOAP message with two entries ......................................................... 38
Table 5. Size of a SOAP message with ten entries ......................................................... 39
Table 6. Size of a SOAP message with twenty entries ................................................... 39
Table 7. Size of a SOAP message with fifty entries ....................................................... 40
Table 8. Size of a SOAP message with one hundred entries .......................................... 40
Table 9: DATT and NAM transfer rates for a Threshold Value calculation period of ten minutes ........................................................................................................................................ 54
Table 10: DATT and NAM transfer rates for a Threshold Value calculated on each transmission ........................................................................................................................................ 54
**Glossary of Abbreviations**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2B</td>
<td>Business to Business</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separated Variable</td>
</tr>
<tr>
<td>DATT</td>
<td>Dynamic Adaptive Threshold Technique</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>EXI</td>
<td>Efficient XML Interchange</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>J2ME</td>
<td>Java 2 Platform, Micro Edition</td>
</tr>
<tr>
<td>MSS</td>
<td>Maximum Segment Size</td>
</tr>
<tr>
<td>NAM</td>
<td>Network Addressable Middleware</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Card</td>
</tr>
<tr>
<td>OPT</td>
<td>One Pass Technique</td>
</tr>
<tr>
<td>PPC</td>
<td>Pocket PC</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>SAX</td>
<td>Simple API for XML</td>
</tr>
<tr>
<td>SGML</td>
<td>Standard Generalised Markup Language</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Orientated Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>TV</td>
<td>Threshold Value</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XHTML Mobile</td>
<td>eXtensible HTML Mobile Profile</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XSL</td>
<td>eXtensible Stylesheet Language</td>
</tr>
<tr>
<td>ZXML</td>
<td>XML Compression Method</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

Extensible Markup Language (XML) [1] derives from SGML [2], which is an ISO standard (ISO 8879). It uses plain text to encode a hierarchical set of information using verbose tags to allow the XML document to be understood without any special reader. XML also allows the use of Schema documents [3] and Namespaces [4] to create a well defined contract describing what a single XML document means and to what it applies.

The self-contained nature of XML and the strong contract provided by its schemas has made XML a very important data format. It is used as a way to store and transport information, as a way of preparing data for publishing [5], translating to another language or structure [5] and even as a message layer between computer systems [6]. It is increasingly finding itself used in a number of different business markets, such as managing financial communications [7, 8] and XML is often considered a useful tool for building internationalized business languages to allow better communication across vendors, platforms and languages [9].

XML is often used on Internet and intranet facing webservers to request and deliver web services used in wider applications. This is seeing wider adoption through the development of Enterprise Service Buses (ESBs) [10] to take advantage of the Service Orientated Archetectures (SOAs) [11] that are being implemented in many organisations. To make matters worse SOAs wrap XML document requests and responses within a Simple Object Access Protocol (SOAP) envelope describing what the XML document is for and detailing any errors that may have occurred during the SOA operation [6, 12, 13].

The devices using XML have also been expanding to include a broad range of newer devices, such as ubiquitous computing devices like Radio Frequency Identification (RFID) [14, 15] and Pocket PCs (PPC). These devices often use languages such as Java 2 Micro Edition (J2ME) [16] to parse XML [17]. This is used to communicate between
the devices and the outside world, often over a wireless network connection. This devices often need to work in a real-time or near real-time manner, servicing a request as soon as it is made. An example of this would be with a RFID tagged stock item passing though a scanner.

XML is also being used to transfer information to wireless devices such as smartphones [18] using the eXtensible HTML Mobile Profile (XHTML Mobile), a combination of XML and HTML [19, 20]. This allows a single source of information to provide content for both a mobile platform, such as smartphones, and in a more traditional form, such as on a web browser in a PC.

XML is even being examined to determine if it can pass rich details regarding a user’s access privileges to data sources, reducing the need for web based applications to look up a user’s details in an Access Control, allowing for lower server overheads [21].

With all of the applications that XML is being used for it is taking up an increasing volume of network traffic, and this was expected to rise to about 25% of network traffic in 2006 from almost 0% in 2002 [22], as shown in Figure 1.

![Figure 1: Increase in XML usage](image)

The increase in XML traffic impacts the availability of the network to other forms of transmission. This will require some level of control on the XML document
transmissions to ensure a certain Quality of Service (QoS) [23] for the whole network.

The increasing popularity of XML and the need to use it in near real-time processes means that we need to make sure the format is as efficient as possible. One of the greatest problems with the efficiency of using XML documents is the size of the plain text structure they use, resulting in longer network transmission times.

There has been a number of methods suggested to reduce the size of XML documents. Examples of these methods include replacing the names of elements used in the XML document, such as with the XML Compression (ZXML) method [24], using a binary representation of the XML document [25, 26, 27], compressing the XML document [28, 29, 30], and using some kind of middleware to control when an XML document is compressed [31, 32].

In this thesis we shall highlight our own methods and middleware solution designed to control the compression of an XML document to be transmitted across a network [33, 34, 35, 36].

1.1 Motivation

We were motivated to find a way to improve the inefficient size of XML documents used to transmit information across a network. We wanted to determine an efficient method to control the size of XML documents that are transmitted across a network to reduce the total time it takes to transmit a document. We also wanted to develop a systematic method to control, with our conditions, the XML document when it is to be transmitted.

We started by investigating the different means of controlling how XML documents are transmitted across a network. This involved a literature review to determine what work has already been done in this area. The next objective was to determine if there are any alternative methods to those already examined that could be used to control how XML documents behave. Having determined a possible new method the third objective was to
develop a program to implement it in a network. Once the program was developed it was tested in several stages, using simple tests to see if the new method holds any value, seeing how it works in an (assumed) unchanging network environment, how it works in a simple network environment where the program is allowed to react to changes in the network and finally using long running tests in the simple network environment. The program will be compared to an existing popular method to see if the new method offers any benefits.

1.2 Thesis Argument

XML is now widely used in business networks to integrate applications and transport data. This thesis looks at how the intensity of this network traffic can be controlled. We look at ways of doing this that are simpler than existing methods and, most importantly, at a method that can be implemented without needing to change applications businesses are already using and that is therefore able to be deployed into existing environments with a minimum effort. The level of isolation from applications and the network tier itself allows the new method to be more easily monitored by network administrators.

This thesis seeks to determine the effectiveness of using compression to improve the transmission times of XML documents passing over a network. Further it aims at developing a technique to aid in the transmission of XML documents by controlling when the XML documents are compressed based on network conditions. We hope that such a technique will allow XML documents of a range of sizes to be transferred across a network faster than if they were all being transmitted as compressed or as uncompressed XML documents.

1.3 Limitations of the study

While this study looks at the use of compression to reduce the transmission time of XML documents, only a few methods of compression, such as GZip and Xmill, were considered at this stage and this study did not consider in depth the affect of using a
tweaked compression algorithm to better reduce the size of the XML document. Another limitation in the study is that only a simple network consisting of two computers, a server and a client, connected by a router was used to test how well the methods discussed perform. While this simple network had its characteristics changed by varying the network activities and the NIC throughputs it does not show the full range of network loadings that are expected in a business network.
Chapter 2
About XML Encoding and Transmission

XML presents a number of problems to businesses. The open, rich nature of the XML documents presents problems with both the size of the XML documents and how they are used in secure environments.

2.1 Encodings used with XML

XML was created as a bridge between the Standard Generalised Markup Language (SGML) and the much simpler HyperText Markup Language (HTML). Like SGML, XML itself is not a markup language but rather it is a specification that allows markup languages to be devised.

XML had a number of simple design goals, and these are listed within the XML specification [37] as:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.
Of these design goals the two most important for what we examined are number 6, that
the document should be human-legible and number 10, that the terseness of the XML
document is of minimal importance.

To make XML human readable it uses a plain text encoding. In general this encoding is
either the Unicode UTF-8 or UTF-16 encodings [38]. Other encodings are allowable for
an XML document, however, support for these encodings is not guaranteed in the XML
Parser that reads the document and in practice XML parsers only use a limited number of
different encodings.

Plain text encodings are not as efficient as other encoding forms, such as binary
encodings. When combined with the goals of making the text reasonably clear and with
terseness not a consideration this means that most XML documents use large descriptive
tags to describe their contents. This will normally lead to documents that take up a lot
more space than they would if they had been encoded without tags that describe the
contents.

2.2 Security and Access control

As XML uses plain text encodings it presents a problem in how to successfully pass
secure information. The obvious solution to this problem is to encrypt the XML
document before passing it. However, the rich content of the XML document allows it to
be used in additional ways.

One such way is suggested by G Miklau and Dan Suciu [21]. They suggest a way in
which database access control information can be passed alongside the information of
interest to consumers. By passing along this access control information they remove the
need for a server to grant such permissions and allow users to interact with the
information offline.
Even though this topic is very important for the transmission of XML documents, we shall put it aside for this thesis and focus on how we can transmit these XML documents efficiently.

2.3 Network Characteristics

Because we shall focus on the behaviour of networks for the transmission of XML documents we first need to examine a number of network characteristics. These characteristics are described here.

When we focused on connectivity on a network we were interested in how XML is transmitted across a network. More specifically, given XMLs use over Internet connections, we were interested in how XML is transmitted across a TCP/IP [39] connection. It is well known that the most important parameters of any network are its bandwidth, throughput, buffer size and latency, which are also used to assess a network.

TCP/IP splits the data being sent across the network into a series of data packets. These packets are then transmitted across the network separately. TCP/IP uses a congestion window to determine how many packets it is able to send at once across the network. This congestion window starts with a low transmission rate of one segment, as described by the receivers Maximum Segment Size (MSS) [39]. This rate is then doubled each time the packets are successfully sent until the transmission rate reaches the receivers maximum allowable rate, that is, the data being transmitted fills the receiver’s buffer size, though obviously the sender’s buffer size will need to be large enough to contain the data to be transmitted or network congestion causes it to stop increasing.

The network’s bandwidth limit lists its maximum theoretical throughput. An Ethernet network connection rated at 100Mbps could theoretically move data across the network at 100 Mbps. The real world speeds are, however, often much lower than this theoretical limit.
The latency of a network describes how long a data packet takes to move from one end of the network to the other or, more formally, it is the time between initiating a request for data and the beginning of the actual data transfer. In a network this occurs as a data packet is stored, analysed and transferred. [40]

A network’s throughput describes how much data can be passed across a network per unit time. For a TCP/IP network this is a function of the buffer size and the network latency, or in other words the maximum throughput is defined as [41, 42]

\[
\text{Throughput}_{\text{max}} = \frac{\text{buffer size}}{\text{latency}}.
\]

Network congestion and other factors may lead to individual data packets not arriving at the receiving destination. This requires the lost packets to be retransmitted and leads to a real world measured throughput that is lower than the maximum value calculated here.

### 2.4 Usage of XML in Business to Business processes

It is well known that XML has been actively involved in Business to Business (B2B) processes. We briefly examine B2B in this section.

B2B transactions are transactions that take place between separate businesses or enterprises. These transactions may include purchasing or the offering of services. These types of interactions are becoming increasingly important in real world applications. ESBs are often being deployed in organisations to allow services to be called both internally and from external organisations, and these interactions need to be efficient.
In order to allow one business to talk to another business to request a product or a service they need to have a common language. XML is rapidly becoming the default language used in these situations [43].

XML schemas [3] describe how a service may be called and what it’s response will look like. For example, one business may send an XML document to another business that complies with the agreed schema to perform a search on client data held by the second business. The second business then sends the client data back to the first as XML complying with the results schema. This means that the first business is able to perform a search and read results from a service owned by the second business.

As these B2B services can potentially carry large volumes of information the XML documents they transmit can be very large. As an example a collection of data on a company’s available products can consume hundreds of kilobytes, or even several megabytes in an XML document. This becomes one of the major problems in using XML driven applications in B2B.
Chapter 3
Current Efficient Methods for XML Transmission

XML is a very useful format; however, its size becomes a barrier when it is used to transmit information across a network. This chapter looks at some of the existing methods for improving the performance of XML across a network by reducing its size.

Section 3.1 will talk about how the names of elements can be moved to an index in order to reduce the size of the document. In Section 3.2 it will describe how an XML document can be written as a binary object to reduce its size. The use of compression on an existing XML document will be discussed in Section 3.3 and in Section 3.4 it will introduce a method of controlled compression, only compressing the XML document when it would be of benefit to do so.

3.1 Element Name Replacement

One way to reduce the size of an XML document is to reduce the amount of repeated information. An XML document contains a number of repeating element tags. Every time an element is included in an XML document its name is present twice, once for an opening element tag and once for a closing element tag. Moreover, in XML documents that use a large number of repeating tags, such as an XML document that is used to pass a set of records from a database, which can be repeated a huge number of times throughout the document. One way of reducing this repetition is to use eXtensibly Stylesheet Language (XSL) to replace the element names in the XML document and reintroduce them when reading the XML document. This method is named the XML Compression Method and is referred to as the ZXML method [23].

An index element is added to a header in the ZXML document by using a compatible parser. This element contains a map of element names and short values, preferably only
one character in length. When the element then appears in the document the name used in the tag is replaced by the index value. The result is a smaller XML document that still complies with the XML standard, having removed the duplicated text from the tags of element names that are used more than once in an XML document.

The problem with this method is that it relies on either having a compatible parser available to create the index and then read back the original information, or it requires applications that make use of this technique to implement additional logic to handle the index. It also requires a character to separate different element values, which needs escaping if that character would be part of a valid element name.

### 3.2 Efficient XML Interchange

The Efficient XML Interchange (EXI) working group was formed by the World Wide Web consortium to look at efficient encodings of XML documents that cause minimal disruption with existing parsers and XML implementations [25].

The EXI working group was formed in 2006 to look at ways in which different encodings can be improved with regard to compactness and processing efficiency. The working group is looking at a number of different methods to improve XML encoding, however to my best understanding the group is still investigating how best to encode XML.

Amongst the encodings being investigated is the Efficient XML encoding [26], which itself implements XML Binary Characterisation [27].

XML Binary Characterisation identified a number of use cases that must be possible when using a binary encoding of an XML document. These use cases describe a number of must support and must not prevent criteria presented in Table 1.
Table 1. XML Binary Characterisation requirements [27]

<table>
<thead>
<tr>
<th>Property</th>
<th>W3C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MUST Support</strong></td>
<td></td>
</tr>
<tr>
<td>Directly Readable and Writable</td>
<td></td>
</tr>
<tr>
<td>Transport Independence</td>
<td>W3C</td>
</tr>
<tr>
<td>Compactness</td>
<td></td>
</tr>
<tr>
<td>Human Language Neutral</td>
<td>W3C</td>
</tr>
<tr>
<td>Platform Neutrality</td>
<td>W3C</td>
</tr>
<tr>
<td>Integratable into XML Stack</td>
<td>W3C</td>
</tr>
<tr>
<td>Royalty Free</td>
<td>W3C</td>
</tr>
<tr>
<td>Fragmentable</td>
<td></td>
</tr>
<tr>
<td>Streamable</td>
<td></td>
</tr>
<tr>
<td>Roundtrip Support</td>
<td></td>
</tr>
<tr>
<td>Generality</td>
<td></td>
</tr>
<tr>
<td>Schema Extensions and Deviations</td>
<td></td>
</tr>
<tr>
<td>Format Version Identifier</td>
<td></td>
</tr>
<tr>
<td>Content Type Management</td>
<td>W3C</td>
</tr>
<tr>
<td>Self Contained</td>
<td></td>
</tr>
<tr>
<td><strong>MUST NOT Prevent</strong></td>
<td></td>
</tr>
<tr>
<td>Processing Efficiency</td>
<td></td>
</tr>
<tr>
<td>Small Footprint</td>
<td></td>
</tr>
<tr>
<td>Widespread Adoption</td>
<td></td>
</tr>
<tr>
<td>Space Efficiency</td>
<td></td>
</tr>
<tr>
<td>Implementation Cost</td>
<td></td>
</tr>
<tr>
<td>Forward Compatibility</td>
<td></td>
</tr>
</tbody>
</table>
Efficient XML reads from an in-memory XML structure, such as the Simple API for XML (SAX) or the Document Object Model (DOM) structure, and directly compresses this to generate a compact XML document. The process of compressing and decompressing the compact XML document would be the responsibility of the parser and implemented in such a way that it does not require an additional stage beyond generating the in-memory structure.

3.3 File Compression

The simplest and most obvious way of improving the transmission of an XML document across a network is to reduce its size by compressing the document before it is sent. This can be done by any number of existing compression technologies, such as the GZip library or commercial compression utilities such as WinZip [28] or WinRar [29].

The plain text nature of the XML documents lends itself very well to the use of compression and it can lead to a massive reduction in the document size, as we will show in Chapter 5. It is possible to further improve the compression ratio of the document by rearranging the order of data within the XML document to group like domains of information. This is the method used by the XMill compressor [30] to further reduce the size of the compressed document.

If we want to use compression to reduce the size of XML documents before we transmit them we require a method of compressing the XML document before it is transmitted as well as a means of decompressing it at the receiver. If we are using a HyperText Transfer Protocol (HTTP) server that supports compression it could be applied automatically if it sees that the receiver also supports compression. However, if a different network technology is used compression needs to be handled by the implemener.
Compression is not without cost as the additional processing required to compress and decompress the document takes additional time and may in fact cause the XML document to take longer to transmit.

### 3.4 Network Adaptable Middleware (NAM)

It’s noted that there is another solution to this problem titled Network Adaptable Middleware (NAM) [31, 32]. This solution estimates the time that it will take for the whole process, that is to compress, transmit in binary format and decompress a document, compared to the time that it would take to transmit the document without compressing it. These estimates are based on a persistent collection of information on how the system has performed in the past and provides a reasonably accurate estimate of whether it would be faster to compress the document before transmission or not.

Whenever NAM transmits an XML document it uses records of the times taken for different events to estimate how long such events are expected to take for the current XML document. NAM then decides if it should send the XML document compressed as a result of the comparison, which can be expressed as:

\[
\text{If } t_{\text{Uncompressed Transmission}} > t_{\text{Document Compression}} + t_{\text{Compressed Transmission}} + t_{\text{Document Decompression}} \\
\text{Then transmit\_compressed,} \\
\text{Else transmit\_uncompressed.}
\]  

where \( t \) is time.

So if the time taken to transmit the XML document uncompressed is less than the time taken to compress, transmit and then decompress the document, the XML document will be sent uncompressed.
As the estimate must be calculated whenever a document is to be sent it is the case that NAM can spend a significant amount of time in determining if the document should be compressed or not. Additionally, as the estimates are based on the size of the XML document being transmitted, NAM assumes that when two different XML documents with the same physical size in bytes are compressed the result will also have the same physical size.
Chapter 4
Dynamic Adaptive Threshold Technique (DATT) Formatting

This study develops and puts forward a technique that could offer a benefit in comparison to existing techniques. We define the concept of a switch that controls the two possible transmission states, to send the XML document compressed, or to send it uncompressed. This switch will change the transmission state based on the size of the XML document and what it knows about the network through a Threshold Value described in Section 4.1. This allows for control to be provided over the size of an XML document at the time of transport without requiring a change to the behaviour of the XML parser or requiring a non-human readable encryption.

Looking at the existing methods presented in Chapter 3 it can be seen that NAM seems to be using a decision based on a switch similar to what is planned. The NAM technique’s decision process on when to compress an XML document appeared to be best able to deal with situations where the computational resources required compressing and decompressing the document caused the overall file transfer to be slower with a compressed document than with an uncompressed document. However the computational resources required in making the decision could make it less efficient. We wondered if the same principles as NAM could be used to transfer XML documents across a network without requiring an expensive calculation based on the historical performance of the network to be performed every time a file was transferred. We anticipate that with a simpler calculation the overall transfer time would be lower than that in NAM for a given range of network behaviour.

This chapter starts by looking at the theory underlying the technique. We start by examining how such a technique would work in a static network environment before we look at how we can have accurate calculations in a changing network. Once we have established our theory we detail our testing of the technique. We start by testing that the reduction in XML document size is significant and worth further investigating. Once we
have established that compression shows valuable size reductions we test how our technique works in short duration tests on a small network made as static as we can get it. Finally we look at how our modified technique works in long runs on a small network.

The proposed model, DATT, is a novel extension to the ideas behind existing methods.

4.1 The Theory of DATT

4.1.1 Using a Threshold Value to decide if Compression is needed.

We wanted to look at a technique that had the same advantages as NAM in allowing a decision process to compress the XML document when it would give an advantage in transfer time, but we also wanted to remove the expensive performance estimate calculation NAM uses. It is well known that large XML documents (such as a large data load document) will always need compression to reduce their transmission time while very small documents (such as a very simple build document) will never need it. As these types of documents would typically make up the bulk of XML traffic it would make sense to not have to perform a complicated calculation every time one was transmitted. My use of the terms large XML document and small XML document should be noted here. The question we have to answer is how much larger does an XML document need to be before it can clearly be said that it should be compressed, and conversely how small does an XML document need to be to say it should never be compressed. One way of determining this is to replace the calculation with a much simpler Threshold Value test to compare the size of the XML document to the size of a document known to take as long to transmit uncompressed as it would take to transmit compressed.

The Threshold Value (TV) is defined as the average file size where an uncompressed XML document will take as long to transmit across a network as it takes a compressed XML document to be compressed, transmitted and then decompressed. This is to say for average network conditions an XML document of a particular size will be transmitted as:
where \( t \) is time, \( UC \) is Uncompressed Transmission, \( DC \) is Document Compression, \( CT \) is Compressed Transmission and \( DD \) is Document Decompression.

To determine the Threshold Value a number of XML documents of various sizes need to be transmitted across a network both in a compressed and uncompressed form. We use Curve fitting on the data points collected to determine the predicted transmission time for both compressed and uncompressed XML documents as the size changes. An example of how curve fitting can be applied to the sets of compressed and uncompressed data to determine the Threshold Value is given in Figure 2.

![Example of using curve fitting to calculate a Threshold Value](image)

Figure 2: Curve fitting example sets of uncompressed and compressed transmission data to determine the Threshold Value.
The intersection of these two lines shows the size of an XML document that will take as long to transmit uncompressed as it does compressed. This value will be termed the Threshold Value. We use a simple linear function when curve fitting as given below:

\[ t = a \times s + b \]  

where \( t \) is the time taken in transmitting the XML document, \( a \) and \( b \) are the coefficients and \( s \) is the size of the XML document.

We perform this curve fitting once each for all the historical data for sending the XML document compressed and for sending it compressed. We can then solve for when the time is expected to be the same when sending the XML document compressed or uncompressed, such that

\[ t_{UT} = a \times s + b = t_{CT} = c \times s + d \]

where \( t_{UT} \) is the total time it takes to transmit the XML document uncompressed, \( t_{CT} \) is the total time it takes to send the XML document compressed, \( s \) is the size of the XML document, \( a \) and \( b \) are the coefficients of the uncompressed XML document transmission and \( c \) and \( d \) are the coefficients of the compressed XML document transmission. Therefore we have

\[ TV = (d-b)/(a-c) \]

where \( a \) and \( b \) are the coefficients determined when curve fitting the uncompressed time, \( c \) and \( d \) are the coefficients for transmitting the compressed XML documents and \( TV \) is the Threshold Value size of an XML document that takes as long to send compressed as it does to send uncompressed.

Once we have calculated the Threshold Value for a network we just need to compare the Threshold Value to the size of the XML document we want to send and to check if it
should be compressed before it is sent. All XML documents with a size that is smaller than Threshold Value are sent uncompressed while all XML documents with a size larger than Threshold Value are sent after being compressed. While XML documents that are the same size as Threshold Value can either be sent with compression or without it, we send them as uncompressed documents for the sake of simplicity.

4.1.2 A Dynamic Threshold

Testing the size of an XML document against a Threshold Value is a very simple calculation to perform and it can easily outperform the far more complicated NAM technique. However the Threshold Value is very dependent on the initial conditions of a network. There are five network factors that contribute to the latency time of delivering a query output [31] based on the analysis of the one gigabyte TPC-H benchmark [44]. We see that as the load on a network changes with traffic, or as the available bandwidth changes or one of the other variables affects the network the initial Threshold Value that was calculated will lose accuracy. While it will still remain a faster way of transferring files than does NAM for a less accurate Threshold Value owing to the simplicity of its calculation, if the network changes are large enough (say at peak network usage times, or in a network of highly variable usage) it will no longer be as effective as NAM. At such large deviations the Threshold Value based calculations will be sending documents that should be compressed as uncompressed documents, or sending documents that should not be compressed as compressed documents.

In order to keep the Threshold Value valid for a network over time it is necessary to periodically re-perform the calculation to determine the Threshold Value accordingly. Recalculating the Threshold Value periodically is a more expensive calculation than the one used to determine if the XML document should be transmitted compressed or not as described in Section 4.1.1. Therefore we seek to use the longest period between calculations, only performing the calculations when necessary. Once it has been re-calculated the new Threshold Value is used for the current time period.
The periodic recalculation will keep the Threshold Value used close to what the real time Threshold Value is for the instantaneous network traffic load and allows for faster network transfers than achieved with NAM.

We have termed this periodically recalculating Threshold Value technique as the Dynamic Adaptive Threshold Technique (DATT).

To calculate a new Threshold Value we need to keep record of the time period it takes for an XML document to be transferred across a network while it is in use. We are then able to use regression to a curve fit for the transfer times taken by both the compressed and uncompressed XML documents in the same way as was done for the simple, fixed Threshold Value calculation in Section 4.1.1.

The calculation of the coefficients to determine the Threshold Value is expensive, so the new Threshold Value should be calculated as infrequently as possible. In terms of time cost, recalculating the Threshold Value every hour would be better than recalculating it every minute for example. The time period over which the Threshold Value should be calculated, however, will depend on the performance of the network and its stability. A network with a load that is changing rapidly in time will require more frequent calculations than a network that is changing slowly. The optimal rate would aim to keep the calculated Threshold Value close to the instantaneous Threshold Value, but for now the rate for a given network is left as a matter of tuning to be performed by the implementers of such a system.

The worst scenario for a network would be the case that it is changing so much over time that it requires the Threshold Value to be recalculated every time an XML document is transmitted. In such a case we would expect the DATT Threshold Value calculation to be more costly than the NAM decision calculation.
4.2 Testing DATT

System testing plays an important role in any established new system. Our testing of DATT is developed as follows.

First, when we observe the test results of the technique’s performance in this thesis in several stages it is found that compression does give significant reductions in size to an XML document, that OPT provides improvements in performance over short periods of time in a simple two node network and that DATT provides improved performance over long periods of time in a simple two node network. These tests were designed to return quantitative data for statistical analysis [45].

We initially started with a simple examination of how compression affects XML documents, looking to get some idea of the size reduction that could be gained to make sure that compressing an XML document would allow for a faster network transmission [33].

Once we had examined the reductions available from compression (and if they seemed like they would make a measurable difference to the file transfer time) we investigated the setting up of a simple Threshold Value test to see how implementing a decision on whether or not to compress an XML document can help improve the performance of a network transfer. We had compared the results of this test to the results obtained by running NAM alongside it [31, 32, 34, 35].

Finally we implemented DATT to recalculate the Threshold Value periodically. This implementation was compared against NAM [31, 32, 36].
4.2.1 Determining the Gain from Compression

Examining the effect of Compression on XML Documents

Before we started to develop tests to validate the idea of using a Threshold Value to control when a document is compressed, we wanted to make sure that the compressed XML documents themselves were worth looking at. We examined how compression would affect a number of different XML documents of varying sizes. To ensure that these messages are closer to what are seen in real business environments we used a Simple Object Access Protocol (SOAP) envelope [6] to contain the details.

Experimental Setup

We created a number of SOAP XML documents containing data that may be found in a typical data load, starting with a SOAP document that contained an empty body. We then created a number of SOAP XML document that contained a simple content structure. This content structure was used to describe SOAP documents that contained a one, two, ten, twenty, fifty or one hundred instances of this simple structure. The empty SOAP XML document and the SOAP XML document with one instance of the simple data structure are listed in Figures 3 and 4 respectively.

```xml
<?XML version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
XMLns:SOAP-ENC="http://schemas.XMLsoap.org/soap/encoding/"
XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```
Figure 3. An empty SOAP XML document
Figure 4. A SOAP XML document with a single entry

We then recreated these SOAP XML documents as simple comma-separated value (CSV) files holding the same information as shown in Figure 5.
As the SOAP XML documents and the CSV files hold the same information it is easy to compare their sizes when they are uncompressed and when they are compressed using the commercially available compression tools WinZip [28] and WinRAR [29]. This will give an idea of what benefits may be attained by passing XML documents compressed and show if it is possible to get the transmission size of XML documents near to the size of other, less structured text formats.

**Results**

We carried out the experiments with the SOAP XML documents and the CSV files. Table 2 shows the results seen when compressing an empty SOAP message.

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed file</th>
<th>WinZip compressed</th>
<th>WinRAR compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>363 bytes</td>
<td>311 bytes</td>
<td>279 bytes</td>
</tr>
</tbody>
</table>

Figure 5. A CSV file holding a single entry
Table 3 shows the compression of a SOAP message that contains a single entry. Compressing the SOAP message results in a size reduction of about 50%, however it is still significantly larger than the CSV file.

Table 3. Size of a SOAP message with one entry

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed file</th>
<th>WinZip compressed</th>
<th>WinRAR compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>1,136 bytes</td>
<td>523 bytes</td>
<td>564 bytes</td>
</tr>
<tr>
<td>SOAP, per data entry</td>
<td>773 bytes</td>
<td>212 bytes</td>
<td>285 bytes</td>
</tr>
<tr>
<td>CSV</td>
<td>138 bytes</td>
<td>221 bytes</td>
<td>187 bytes</td>
</tr>
</tbody>
</table>

Table 4 shows the compression of a SOAP message with two entries. With two data entries the size cost saving is up to 30%. The CSV file is still significantly smaller, particularly when similarly compressed.

Table 4. Size of a SOAP message with two entries

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed file</th>
<th>WinZip compressed</th>
<th>WinRAR compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>1,878 bytes</td>
<td>622 bytes</td>
<td>574 bytes</td>
</tr>
<tr>
<td>SOAP, per data entry</td>
<td>758 bytes</td>
<td>156 bytes</td>
<td>148 bytes</td>
</tr>
<tr>
<td>CSV</td>
<td>298 bytes</td>
<td>271 bytes</td>
<td>240 bytes</td>
</tr>
<tr>
<td>CSV, per data entry</td>
<td>379 bytes</td>
<td>136 bytes</td>
<td>120 bytes</td>
</tr>
</tbody>
</table>

With the ten data entries seen in Table 5 the size cost saving is now around 18%. The CSV file is actually larger than the compressed SOAP file and the compressed CSV files offer only a small size cost improvement.
Table 5. Size of a SOAP message with ten entries

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed file</th>
<th>WinZip compressed</th>
<th>WinRAR compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>8,384 bytes</td>
<td>1,484 bytes</td>
<td>1,358 bytes</td>
</tr>
<tr>
<td>SOAP, per data</td>
<td>802 bytes</td>
<td>117 bytes</td>
<td>108 bytes</td>
</tr>
<tr>
<td>entry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSV</td>
<td>2,153 bytes</td>
<td>933 bytes</td>
<td>923 bytes</td>
</tr>
<tr>
<td>CSV, per data</td>
<td>215 bytes</td>
<td>93 bytes</td>
<td>92 bytes</td>
</tr>
<tr>
<td>entry</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The twenty data entries shown in Table 6 show the size cost saving is now 14% and there is decreased difference between the compressed SOAP document when compared to the compressed CSV file.

Table 6. Size of a SOAP message with twenty entries

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed file</th>
<th>WinZip compressed</th>
<th>WinRAR compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>16,623 bytes</td>
<td>2,352 bytes</td>
<td>2,175 bytes</td>
</tr>
<tr>
<td>SOAP, per data</td>
<td>813 bytes</td>
<td>102 bytes</td>
<td>95 bytes</td>
</tr>
<tr>
<td>entry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSV</td>
<td>4,572 bytes</td>
<td>1,667 bytes</td>
<td>1,654 bytes</td>
</tr>
<tr>
<td>CSV, per data</td>
<td>229 bytes</td>
<td>83 bytes</td>
<td>83 bytes</td>
</tr>
<tr>
<td>entry</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7 shows the results of a SOAP message with fifty entries. The size cost saving by compressing the SOAP message is down to 11% of the uncompressed size. The
compressed SOAP document is now only about 12% larger than the compressed CSV file.

Table 7. Size of a SOAP message with fifty entries

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed file</th>
<th>WinZip compressed</th>
<th>WinRAR compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>41,426 bytes</td>
<td>4,742 bytes</td>
<td>4,515 bytes</td>
</tr>
<tr>
<td>SOAP, per data entry</td>
<td>821 bytes</td>
<td>89 bytes</td>
<td>84 bytes</td>
</tr>
<tr>
<td>CSV</td>
<td>11,915 bytes</td>
<td>3,853 bytes</td>
<td>3,851 bytes</td>
</tr>
<tr>
<td>CSV, per data entry</td>
<td>238 bytes</td>
<td>77 bytes</td>
<td>77 bytes</td>
</tr>
</tbody>
</table>

The results of one hundred data entries in a SOAP message are shown in Table 8. The size cost saving is still about 11% of the uncompressed SOAP message size. The compressed SOAP document is now only about 9% larger than the compressed CSV file.

Table 8. Size of a SOAP message with one hundred entries

<table>
<thead>
<tr>
<th></th>
<th>Uncompressed file</th>
<th>WinZip compressed</th>
<th>WinRAR compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>83,320 bytes</td>
<td>8,971 bytes</td>
<td>8,798 bytes</td>
</tr>
<tr>
<td>SOAP, per data entry</td>
<td>830 bytes</td>
<td>87 bytes</td>
<td>85 bytes</td>
</tr>
<tr>
<td>CSV</td>
<td>24,708 bytes</td>
<td>7,903 bytes</td>
<td>7,900 bytes</td>
</tr>
<tr>
<td>CSV, per data entry</td>
<td>247 bytes</td>
<td>79 bytes</td>
<td>79 bytes</td>
</tr>
</tbody>
</table>

All the observations show that the efficiency of the compression applied to an XML document improves as more data instances are added. This would be expected given that the structured nature of the XML format means that each data instance would include the same opening and closing element tags. What was interesting is how the compressed SOAP XML documents began to approach the size of the compressed CSV files. As more data instances are added to the SOAP XML document and to the CSV file the difference in their compressed sizes is becoming negligible. The SOAP XML documents richly structured format would make this a preferable format for describing large sets of data.
4.2.2 Using a Simple Threshold Value to determine if Compression is needed – The One Pass Technique (OPT)

**Background of the OPT**

After making sure that there were significant gains to be made using compression on larger XML documents we went on to look at how a simple test could be used to determine when an XML document was large enough to benefit from being compressed.

OPT uses a value for its switch based decision. This value is the size of an XML document that takes as long to transmit as a compressed document over a network as it takes to transmit uncompressed. This value is defined as the Threshold Value. It should be noted that a Threshold Value describes the average traffic situation in a period of time. Therefore we need to determine this Threshold Value by transferring a number of XML documents of different sizes across the network. Transmissions need to be made both with the document compressed, using the XMill compression technology as an example, (and decompressed where it is received) and by transmitting the document without compression.

Once we have obtained the times for the transmission of a range of XML documents we perform a simple curve fitting on the data obtained to determine the Threshold Value. That is the XML document size at which the transmission time for a compressed XML document and an uncompressed XML document would be the same. The Threshold Value is calculated using equation 5 given in Section 4.1.1. If a network is allowing transfers that are particularly fast this may result in a threshold value low enough that all XML documents that are transmitted across it will be uncompressed. Conversely if network transfers are particularly slow all XML documents transferred would be compressed before transmission.

It is important to realise that a Threshold Value will not know the parameters of a network transfer when an XML document is to be transferred, such as changes to the bandwidth and the packet loss rate. Therefore, the Threshold Value will only be valid for
the network bandwidth for which it is calculated. When the bandwidth changes the
Threshold Value currently used will give an inaccurate result. A new Threshold Value
will then need to be calculated. Therefore the Threshold Value is in fact the
instantaneous Threshold Value.

The performance of compression and decompression times is dependent on the CPU
load. If the load running on a CPU is heavier (or lighter) than it was when calculating the
Threshold Value it may not make the appropriate decision on whether or not to use
compression on the XML document. Similarly the technique works best with a
homogenous set of CPUs. The compression/decompression time of two low end CPUs
on a network will be different to the compression/decompression time of two high end
CPUs on the same network using the same Threshold Value. This may lead to the OPT
making a wrong decision on whether or not to compress the document.

OPT can also be affected by changes in the network’s traffic density, when the number of
requests being made across the network either increases or decreases significantly. If the
network is under a heavier load with greater network traffic than it was when the
Threshold Value was calculated the technique is more likely to transmit an uncompressed
XML document when a compressed document would have been faster, and with a lighter
network load compressed XML transmissions are more likely to occur when an
uncompressed transmission would have been faster. It is the case that OPT is supposed
to run in a homogenous environment where the network bandwidth is well known and
network traffic is reasonably stable.

We wanted to compare the OPT to another existing similar techniques and see if it can
offer any advantage. Of the techniques we examined Ghandeharizadeh et al NAM [31,
32] made the closest comparison with the technique we present in this thesis. We
implemented both our own OPT and NAM in the same network environment and
compared the times taken for each method transmitting a number of XML documents in a
range of sizes.
Experimental Setup

We wrote a short Java program to send XML documents from one computer to a second using a TCP/IP connection [39]. This program, presented in appendix B, used two methods to determine if a given XML document should be compressed before it was sent. The first method used OPT. The Java program was given a parameter when it was started to tell it what the Threshold Value for the network was. Once it had received this parameter it would compare the size of the XML document it was going to send to the Threshold Value. If the size of the XML document was larger than the Threshold Value it would compress the XML document before transmitting it. If the XML document was smaller than the Threshold Value, or the same size, it would send the XML document uncompressed.

We also implemented the NAM method in the application. NAM estimates the time it would take to send an XML document in both a compressed state and in an uncompressed one. It uses the statistical package Weka 3.4 [46] to estimate the times taken to compress, decompress and transmit to compressed XML documents as well as how long it would take to transmit the XML document uncompressed. This calculation is done against the set of historical data on previous XML document transmissions NAM has conducted. If the estimation finds that it is faster to transmit the XML document compressed then it compresses the XML document before it is transmitted.

In order to make a fair comparison, the network usage was equally distributed between the tests running OPT and the tests running NAM. The program is alternated between OPT and NAM at runtime by passing it different command line arguments allowing me to easily switch between the two techniques between test runs.

We use two PCs connected together to simulate a network to run these tests. This network is detailed in Figure 6 and consisted of a single client PC (754pin Athlon64 3200+ @ 2.05GHz with 1GB RAM) and a server (Celeron D 2.8@2.79GHz with 512MB
RAM) connected by a Router (Billion BIPAC 7402G) over a 100MBit Ethernet connection.

To conduct the tests we gathered 27 XML documents in a variety of sizes, ranging from 39 Bytes up to 7.3 Megabytes. In order to determine the Threshold Value for use with OPT we transmitted each of the XML documents we were using during the testing over the network both compressed and uncompressed. These results were then used to calculate the Threshold Value using equation 5 from chapter 4.1.1.

As NAM uses existing values to perform its estimates the results found when determining the Threshold Value were also used to provide initial seed values for NAM.

Once this setup was complete we used it to pass each of the 27 XML documents a total of ten times each using both NAM and OPT. As a final test that OPT was working correctly we examined what happened when the Threshold Value was artificially changed to ±10% and ±20% of the Threshold Value we calculated and these new Threshold Values were run in comparison to the calculated one.

We recorded all times in the system, including the time taken to transmit the document, the compression and decompression times (if the XML document was compressed) as
well as the time it took to perform the calculation to see if the XML document should be compressed before being transmitted.

**Results**

We calculated the Threshold Value to be 425 Kilobytes.

Figure 7 presents the results obtained over a number of runs, with the smallest XML documents being transmitted at low run numbers and the larger XML documents transmitted at the higher run numbers. A sample of the data returned during the experimental runs is presented in Appendix C. It is important to note that the largest XML document used in the last run was an order of magnitude larger than the next largest XML document and as such the longer transfer period recorded in the results was expected. More interesting were the results for very small XML documents, typically less than 100 bytes. We noticed a jump in the time it took to transfer these small XML documents when compared to XML documents that were over a kilobyte in size. We are not sure of the reasons for this.

When we compared the transmission times between OPT and NAM we found the results that OPT was able to perform slightly better than NAM. The summed total time taken by OPT to transfer all of the XML documents used in the experimental runs was 148.90656 ms less than the time taken by NAM.
Figure 7: Total NAM transmission time vs. total OPT transmission time.

Figure 8 presents the results we obtained for the time OPT and NAM took to decide if the XML document should be sent compressed or uncompressed. These results use the same XML documents used in Figure 7. It is obvious that the OPT decision time tends to be lower than the decision time taken by NAM, however, both calculations are very dependent on the CPU load and slight changes to this load can greatly affect the time taken to make the decision.

We found that the sum total of the decision times taken to determine if the XML document should be compressed was lower in OPT than in NAM over the period of the experimental runs by 0.43078 ms.
Figure 8 shows the total decision and transfer times for OPT and NAM. The sum total of the times recorded during the experimental runs show that OPT took $149.33734$ ms less time than NAM to transfer the XML documents.

Figure 8: NAM decision time vs. OPT decision time

Figure 9 shows the total decision and transfer times for OPT and NAM. The sum total of the times recorded during the experimental runs show that OPT took $149.33734$ ms less time than NAM to transfer the XML documents.
Figure 10 shows what happens when we deliberately used an incorrect Threshold Value. Using the existing network, which we knew to have a real Threshold Value of 425, we transferred the XML documents using Threshold Values of ±10% and ±20% of the real Threshold Value. These deliberately incorrect Threshold Values were 340 Kilobytes, 382.5 Kilobytes, 467.5 Kilobytes and 510 Kilobytes. We then performed the experimental runs for OPT again, using the incorrect Threshold Values to make the decision on whether to transmit the XML documents compressed or uncompressed.
What we saw was that smaller XML documents would consistently take longer to transmit if they had to be compressed and decompressed (Figure 6). This shows that XML documents larger than the Threshold Value should be compressed before being transmitted while XML documents that are below the size of the Threshold Value should be transmitted uncompressed, with worst case examples taking more than 10 times longer than they should if the decision on whether or not to compress the XML document is incorrect.

Figure 10: Transfer times for a network with Threshold Value 425, using ±10% and ±20% of the Threshold Value for OPT decisions.

We have shown that for a simple, relatively static network a simple Threshold Value calculated for OPT is able to outperform the more complicated NAM calculations. However, networks are not static environments, and even the act of transferring an XML document will perturb one. As such we would not expect the Threshold Value from OPT
to remain a good indicator of when to compress an XML document before sending it as a network’s characteristics change over time and we would expect NAM to outperform OPT in such a changing network. To keep the OPT competitive with NAM we modify it to become DATT.

4.2.3 Extending Simple Threshold Value to the DATT

Background of the DATT

Section 4.2.2 showed that a simple Threshold Value as described by OPT can significantly improve the performance of XML document transmission, with time savings of 149 ms over NAM in the test runs. However, as we discussed in Section 4.1.2 a Threshold Value depends on many factors on the network [31, 44]. When these factors change the Threshold Value calculated for OPT will lose accuracy. If we want OPT to continue working for a current network the Threshold Value must be re-calculated from time to time depending on the current status of the network, that is the Threshold Value must be dynamically calculated to follow the changes on network. This is the basic idea of using Dynamic Adaptive Threshold Transmission (DATT) for controlling XML data transmissions on networks.

DATT recalculates a Threshold Value over a certain fixed period. This period relates to the variability of the network that it is used on. This period would need to be tuned by implementers to make sure the Threshold Value calculation is only being performed when it is necessary. Due to the current research scope we shall look at only two possible cases for the Threshold Value calculation period. The first of these calculates a new Threshold Value after every ten minutes have elapsed. This is because this is a statistically normal period that could be used in a production system. The second period we wanted to examine correlated to every transmission, which means that the Threshold Value will be calculated every time an XML document is to be sent, which relates to the heaviest calculation load scenario for DATT.
As with OPT we compared the DATT method to NAM [31, 32]. We implemented both methods and used a number of XML document transmissions to measure the efficiency of each method.

**Experimental Setup**

DATT required me to develop a short Java programme to check the current conditions of the network as described above. The current Threshold Value is calculated in the same manner as used for OPT, again using the Weka 3.4 [46] package to perform the actual calculation. When DATT calculates a new Threshold Value it will replace the previous one to control when an XML document is transmitted compressed and when it is sent uncompressed. Since the threshold is monitored dynamically the adaptive threshold will always keep a record of the times taken in transferring the data in order to perform the calculations to determine the new threshold.

We used the same network as was used for OPT (Figure 6). Again we used the 27 XML documents in a variety of sizes, ranging from 39 Bytes up to 7.3 Megabytes we had gathered for the OPT experiment in Section 4.2.2. The seed values that had been gathered for the initial NAM calculations in Section 4.2.2 were used again to provide values for the calculations used by NAM as well as calculating the initial Threshold Value for DATT as in Figure 4.

We initially configured DATT to recalculate its Threshold Value every ten minutes. We then ran the experiment using both DATT and NAM on the same network over a period of two days using a CRON job through pycron [47, 48] to trigger the transmission. Once we had this data set we ran the experiment again, getting DATT to recalculate the Threshold Value every time it transmitted an XML document. During these runs the network was being perturbed by running other transfers and downloads.
We recorded all times in the system, including the time taken to transmit the document, the compression and decompression times (if the XML document was compressed), the time it took to perform the calculation to see if the XML document should be compressed before being transmitted, the time it took to calculate the new Threshold Values for DATT and the different Threshold Values used by DATT over the time period.

Results

The results for the decision time, in terms of average for all the runs, for DATT are shown in Figure 11. As the process of determining the threshold has been passed out to a separate process, the decision times for DATT are very short.

![Figure 11: DATT decision time](image)

The results for the decision time, in terms of average for all the runs, for NAM are shown in Figure 12. It is seen that, as NAM accumulates more data upon which to make a decision, the time it takes to actually make a decision increases to allow all the data to be read.
In figures 11 and 12, the average decision times are represented by dashed lines.

When the experiment was performed we stored the results for compressed XML documents separately to the results for uncompressed documents. As we were interested only in the average time taken to make the decision and we were not interested in the order the decisions were made we simply appended the results of the compressed and uncompressed XML documents. It appears that there is a periodic behaviour in Figure 12, with the structure repeating at approximately 500 runs, however, this is only a consequence of appending the two sets of data.

In these experiments, the average decision time for DATT is 0.187622 milliseconds and for NAM is 41.381926 milliseconds, which means NAM takes 220.56 times longer than DATT to make the decision on whether or not to compress the XML document.

It is interesting to note that the experimental results show DATT compressed 587 of 1201 running files, which gives the compressing ratio of 0.488. In contrast to DATT, the NAM compressed 510 of the 1219 running files, which gives a compressing ratio of 0.418. This shows that the NAM is compressing about 86.4% the number of files as DATT. This also shows that, in comparison with DATT, NAM is always (or in terms of the average)
making cautious decisions to keep itself in optimum states but by doing so is causing heavier network traffic, which means that DATT will make higher quality network transfers for XML data on networks. This improvement for DATT compared with NAM is about 0.25%.

Looking at the rate DATT is able to transfer the XML documents using a ten minute period compared to NAM it can be seen that DATT is able to run at about 116% the efficiency of NAM (Table 9)

<table>
<thead>
<tr>
<th>Method</th>
<th>Total data transferred (Bytes)</th>
<th>Total transfer time (Seconds)</th>
<th>Transfer rate (b/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATT</td>
<td>1167003928</td>
<td>217.618</td>
<td>5368690</td>
</tr>
<tr>
<td>NAM</td>
<td>1185040086</td>
<td>255.55</td>
<td>4637210</td>
</tr>
</tbody>
</table>

We also measured what happens when the network is so unstable that DATT needs to calculate a new Threshold Value every time an XML document is transmitted (Table 10). It is seen that in this worst case scenario DATT is running at only about 46% of the efficiency of NAM.

<table>
<thead>
<tr>
<th>Method</th>
<th>Total data transferred (Bytes)</th>
<th>Total transfer time (Seconds)</th>
<th>Transfer rate (b/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATT</td>
<td>1379168005</td>
<td>496.579</td>
<td>2777900</td>
</tr>
<tr>
<td>NAM</td>
<td>1379161357</td>
<td>285.977</td>
<td>4822630</td>
</tr>
</tbody>
</table>

It can be seen that DATT would not provide benefits in network environment with random loads.
Chapter 5
Discussion

We have seen that compression provides a means of significantly reducing the size of an XML document allowing it to be more easily transported over a network, with a large SOAP message having only 11% of its original size when it has been compressed [33]. However, we have also seen that the time it takes to compress and to decompress an XML document can in some cases cause the overall transmission of the XML document to take longer than simply transmitting the XML document uncompressed.

There exists a middleware methodology in NAM [31, 32] that is able to determine when an XML document would benefit from being compressed before it is transmitted over a network. NAM, however, requires an expensive calculation to be performed before it transmits each XML document. We were able to set-up a simple Threshold Value test in the OPT [34, 35] that did not require the expensive calculation from NAM. By compressing only the XML documents that were larger than this Threshold Value we were able to reduce the overall time taken to transmit an XML document when compared to NAM, saving a total of 149.33734 ms. We also found that using an incorrect Threshold Value could lead to a large number of incorrect decisions on whether to compress the XML documents or not. These incorrect decisions could lead to an XML document taking up to ten times as long to transmit compared to the correct decision.

The changing nature of networks will mean that the number of incorrect decisions with OPT are likely to increase with time. Allowing the Threshold Value to be recalculated periodically using DATT [36] keeps the Threshold Value based decisions from gaining too many inaccuracies. This, however, increases the overall processing load as the server now needs to periodically recalculate the Threshold Value. The period at which this Threshold Value needs to be recalculated depends on how rapidly the network environment is changing. When the network is changing greatly from one moment to the next the Threshold Value will need to be calculated frequently. When the network...
environment is changing only slowly, as may be the case in off-peak times, the Threshold Value may need only to be calculated infrequently.

We found that when DATT is using a period of ten minutes it has an efficiency of 116% that of NAM. However, in the worst case scenario of having to calculate the Threshold Value every time an XML document is transferred we find that DATT is operating only at 46% the efficiency of NAM. This would suggest that DATT is a better choice than NAM for relatively stable networks; however, NAM would be better able to handle rapidly changing networks.

One improvement we could make to the DATT methodology presented here would be to move the processor time expensive Threshold Calculation to a different processing thread running on a different processor core or even a different computer. This would then remove the impact that calculating the Threshold Value would have on the network transfers of XML documents.

A further area that might prove productive is in looking at methods to determine the best calculation period to use for a network. At the moment determining this period is a manual tuning process that requires trial and error. A method to determine the value would make setting up a DATT transfer mechanism far faster and would have a good chance of improving the value used for the DATT period.

We also wish to see if DATT could be extended to allow it to control the volume of the XML document traffic. This would be a QoS [23] implementation designed to prevent XML traffic from impacting on other, higher priority traffic.
Chapter 6
Conclusion and Further Work

In this thesis we have identified a number of different ways of improving the efficiency in the size of XML documents for network transport, ranging from removing repeated element tags [24], rendering the XML document as a binary object [25, 26, 27], compressing the XML document [28, 29, 30] and using a middleware solution to selectively switch when an XML document is compressed [31, 32].

This thesis sought to determine the effectiveness of using compression to improve the transmission times of XML documents passing over a network. We have examined the benefits of using a straight compression process to reduce the size of an XML document [33] and we have found that a compressed XML document shows a significant size reduction when compressed. This size reduction increases for XML documents that contain repeating sets of similar information, such as an XML document used to represent a list of books. In these cases the compressed XML document was found to be only 11% the size of the original, uncompressed XML document. This reduction in size leads to a reduction in the time required to transmit the XML document across a network.

Having found that the size of XML documents can be reduced considerably when compressed we wanted to develop a technique that uses compression to reduce the transmission time of an XML document across a network. We needed to take the time taken to compress and decompress an XML document into account when determining the transmission time. We developed the OPT method [34, 35], defining a simple Threshold Value for a network. The Threshold Value describes the point at which it is faster to compress and transmit an XML document than it is to send an XML document uncompressed. The Threshold Value is therefore network dependent and needs to be determined before OPT can be used.

Comparing OPT to NAM in a simple, slowly changing network environment over short
periods transferring 27 XML documents ten times each we found that OPT was able to perform the XML document transfers 149.33734 ms faster than NAM.

We examined OPT to see what happens when the Threshold Value is incorrect for a network. We found that an incorrect Threshold Value increased the number of times an XML document was sent uncompressed when it should have been compressed and vice versa. These incorrect decisions lead to the XML document taking ten times longer to transfer in the worst cases.

As networks are not static we developed DATT to change the Threshold Value over time, keeping it in sync with the network and able to continue making the correct decision on whether to compress the XML document or not [36]. We compared DATT to NAM over extended runs to see how they compared. During these runs DATT was calculating the new Threshold Values in the same process as it was performing the calculations. We found that when DATT was performing the Threshold Value calculations every ten minutes DATT was providing 116% the efficiency of NAM. However, in the worst case scenario, when DATT was performing the Threshold Value calculation before transmitting an XML document, DATT was providing only 46% the efficiency of NAM. Our results show that NAM is a good choice for networks with traffic loads that are changing in a very rapid manner; however, for networks with a regular load DATT becomes favoured.

We have shown that compression of XML documents before they are transmitted can reduce the time it takes to transmit them across a network. Further we have developed a new technique, DATT, that allows compression to be controlled to minimise the time taken to transmit an XML document across a network and that this new technique is able to outperform existing popular techniques in conditions where the traffic load on a network is not changing rapidly.

Future directions of study of DATT will examine how it can be modified to provide a level of QoS for XML documents running on a network and will aim at determining how
XML document traffic can be controlled when other, more time critical network traffic such as streaming video for conferences.

We would also like to examine how security controls can be introduced to the DATT to allow XML documents to be transmitted over networks without the need for external controls or encryption.
Bibliography


Appendices

Appendix A: My Contributions


Appendix B: Program Code

In order to test the usefulness of OPT a program was written to transmit XML documents across a network. This program consisted of a class to transmit the XML documents using OPT to determine if it should be sent as a compressed or as an uncompressed document and a second class to use NAM to determine if it should be sent as a compressed or as an uncompressed document. Various support classes were used to carry out required calculations and a UDP server was created using Sun's UDPT imeClient. The application was run as a scheduled CRON job, executing the OPT and NAM classes alternatively and periodically executing the ThresholdCalculator to recalculate the Threshold Value to use with OPT.

RunApplication

/**
 * This class is used to pass an XML document over a network to a
 * server using OPT to determine if the file should be sent compressed or uncompressed.
 */
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.net.Socket;
import util.FileToBytes;
import util.GZipDocumentCompression;

public class RunApplication {

/**
 * default values size: 100 address: 192.168.1.2 port: 9013
 */

private static final int BUFSIZE = 32; // Size of receive buffer

public static void main(String[] args) {
    // TODO Auto-generated method stub
    String address = args[0];
    int port = Integer.parseInt(args[1]);
    int compressed = 0;
    int loopCounter;
    long startCompress = 0;
    long endCompress = 0;
}
long startTransfer;
long endTransfer;
long transferTime;
long startCalculation;
long endCalculation;
long calculationTime;

// Used to read in the decompress time
byte[] byteBuffer = new byte[BUFSIZE];

// The file is passed by the args.
File selFile = new File(args[2]);
long fileSize = selFile.length();

// Single file to collect all compressed results into one place.
FileOutputStream logCompressedOut;
String logCompressedOutPath = "C:/Temp/Compressed.txt";
// Single file to collect all uncompressed results into one place.
FileOutputStream logUncompressedOut;
String logUncompressedOutPath = "C:/Temp/Uncompressed.txt";

// Single file to hold the threshold value.
BufferedReader thresholdFile;
String thresholdFileName = "C:/Temp/threshold.txt";
try {
    logCompressedOut = new FileOutputStream(logCompressedOutPath, true);
    logUncompressedOut = new FileOutputStream(logUncompressedOutPath, true);
    thresholdFile = new BufferedReader(new FileReader(thresholdFileName));
    long thresholdValue = new Double(thresholdFile.readLine().trim()).longValue();
    String filePath = selFile.getAbsolutePath();
    selFile = new File(filePath);
    String originalFile = filePath;
    System.out.println(selFile.getName());
    selFile = new File(originalFile);
    filePath = originalFile;
    startCalculation = System.nanoTime();
    boolean compress = selFile.length() > (thresholdValue);
    endCalculation = System.nanoTime();
    calculationTime = endCalculation - startCalculation;
    if (compress) {
        System.out.println("Starting to compress document ");
        startCompress = System.nanoTime();
        // GZip compression
        GZipDocumentCompression.compress(filePath);
        endCompress = System.nanoTime();
        filePath = filePath.substring(0, filePath.length() - 1) + "i";
        selFile = new File(filePath);
        compressed = 1;
        System.out.println("Finished compressing document ");
    }
try {
    byte[] fileBytes = FileToBytes.getBytesFromFile(selFile);

    // Create socket that is connected to server on specified port
    Socket socket = new Socket(address, port);
    socket.setKeepAlive(true);
    System.out.println("Connected to server...sending echo string");

    InputStream in = socket.getInputStream();
    OutputStream out = socket.getOutputStream();

    startTransfer = System.nanoTime();
    out.write(compressed);
    out.write(fileBytes); // Send the encoded string to the server
    in.read();
    endTransfer = System.nanoTime();
    in.read(byteBuffer);
    String decompressTime = new String(byteBuffer);
    socket.close(); // Close the socket and its streams

    transferTime = endTransfer - startTransfer;

    if (compressed == 1) {
        logCompressedOut.write("\n".getBytes());
        logCompressedOut.write(new Long(fileSize).toString()
                .getBytes());
        logCompressedOut.write("",".getBytes());
        logCompressedOut.write(new Long(calculationTime).toString()
                .getBytes());
        logCompressedOut.write("",".getBytes());
        logCompressedOut.write(new Long((endCompress - startCompress)).toString()
                .getBytes());
        logCompressedOut.write("",".getBytes());
        logCompressedOut.write(new Long(transferTime).toString()
                .getBytes());
        logCompressedOut.write("",".getBytes());
        logCompressedOut.write(new Long(decompressTime).getBytes());
        // As we are including the Threshold time calculation with
        // each pass, record the time it takes.
        logCompressedOut.write("",".getBytes());
    } else {
        logUncompressedOut.write("\n".getBytes());
        logUncompressedOut.write(new Long(fileSize).toString()
                .getBytes());
        logUncompressedOut.write("",".getBytes());
        logUncompressedOut.write(new Long(calculationTime)
                .toString().getBytes());
    }
}
logUncompressedOut.write(";", getBytes());
logUncompressedOut.write(new Long(
        (endCompress - startCompress)).toString()
        .getBytes());
logUncompressedOut.write(";", getBytes());
logUncompressedOut.write(new Long(transferTime).toString()
        .getBytes());
logUncompressedOut.write(";", getBytes());
logUncompressedOut.write(decompressTime.getBytes());
// As we are including the Threshold time calculation with
// each pass, record the time it takes.
logUncompressedOut.write(";", getBytes());
} catch (IOException e) {
    e.printStackTrace();
    return;
} finally {
    if (compressed == 1) {
        selFile.delete();
        compressed = 0;
    }

    System.out.println("Transfer complete");
}

} catch (Exception e) {
    System.out.println("Unable to create log file.");
    System.out.println(e.getMessage());
    System.out.println(e.getStackTrace());
}
System.exit(0);
RunNam

/**
 * This class is used to pass an XML document over a network to a
 * server using NAM to determine if the file should be sent compressed or uncompressed.
 */
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.Array;
import java.net.Socket;
import util.FileToBytes;
import util.GZipDocumentCompression;

public class RunNam {

    private static final int BUFSIZE = 32; // Size of receive buffer

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String address = args[0];
        int port = Integer.parseInt(args[1]);
        int compressed = 0;
        int loopCounter;
        long startCompress = 0;
        long endCompress = 0;
        long startTransfer;
        long endTransfer;
        long transferTime;
        long startCalculation;
        long endCalculation;
        long calculationTime;

        // Used to read in the decompress time
        byte[] byteBuffer = new byte[BUFSIZE];

        // The file is passed ny the args.
        File selFile = new File(args[2]);
        long fileSize = selFile.length();

        // NAM Calculation
        calculateTimes(selFile.length());

        // Single file to collect all compressed results into one place.
        FileOutputStream logCompressedOut;
        String logCompressedOutPath = "C:/Temp/NamCompressed.txt";

        // Single file to collect all uncompressed results into one place.
        FileOutputStream logUncompressedOut;
        String logUncompressedOutPath = "C:/Temp/NamUncompressed.txt";
    }
}
try {
    logCompressedOut = new FileOutputStream(logCompressedOutPath, true);
    logUncompressedOut = new FileOutputStream(logUncompressedOutPath, true);

    String filePath = selFile.getAbsolutePath();
    selFile = new File(filePath);
    String originalFile = filePath;
    System.out.println(selFile.getName());
    selFile = new File(originalFile);
    filePath = originalFile;

    // startProcess = System.nanoTime();
    startCalculation = System.nanoTime();
    // NAM Calculation
    boolean compress = calculateTimes(selFile.length());
    endCalculation = System.nanoTime();
    calculationTime = endCalculation - startCalculation;
    if (compress) {
        System.out.println("Starting to compress document "+ selFile.getName());
        startCompress = System.nanoTime();
        // GZip compression
        GZipDocumentCompression.compress(filePath);
        endCompress = System.nanoTime();
        filePath = filePath.substring(0, filePath.length() - 1) + "i";
        selFile = new File(filePath);
        compressed = 1;
        System.out.println("Finished compressing document "+ selFile.getName());
    }
}

try {
    byte[] fileBytes = FileToBytes.getBytesFromFile(selFile);

    // Create socket that is connected to server on specified port
    Socket socket = new Socket(address, port);
    socket.setKeepAlive(true);
    System.out.println("Connected to server...sending echo string");

    InputStream in = socket.getInputStream();
    OutputStream out = socket.getOutputStream();
    startTransfer = System.nanoTime();
    out.write(compressed);
    out.write(fileBytes); // Send the encoded string to the server
    in.read();
    endTransfer = System.nanoTime();

    in.read(byteBuffer);
    String decompressTime = new String(byteBuffer);
}
socket.close(); // Close the socket and its streams

transferTime = endTransfer - startTransfer;

if (compressed == 1) {
    logCompressedOut.write("\r\n".getBytes());
    logCompressedOut.write(new Long(fileSize).toString().getBytes());
    logCompressedOut.write(";".getBytes());
    logCompressedOut.write(new Long(calculationTime).toString().getBytes());
    logCompressedOut.write(";".getBytes());
    logCompressedOut.write(new Long((endCompress - startCompress)).toString().getBytes());
    logCompressedOut.write(";".getBytes());
    logCompressedOut.write(new Long(transferTime).toString().getBytes());
    logCompressedOut.write(";".getBytes());
    logCompressedOut.write(decompressTime.getBytes());
} else {
    logUncompressedOut.write("\r\n".getBytes());
    logUncompressedOut.write(new Long(fileSize).toString().getBytes());
    logUncompressedOut.write(";".getBytes());
    logUncompressedOut.write(new Long(calculationTime).toString().getBytes());
    logUncompressedOut.write(";".getBytes());
    logUncompressedOut.write(new Long((endCompress - startCompress)).toString().getBytes());
    logUncompressedOut.write(";".getBytes());
    logUncompressedOut.write(new Long(transferTime).toString().getBytes());
    logUncompressedOut.write(";".getBytes());
    logUncompressedOut.write(decompressTime.getBytes());
}

} catch (IOException e) {
    e.printStackTrace();
    return;
} finally {
    if (compressed == 1) {
        selFile.delete();
        compressed = 0;
    }
}

System.out.println("Transfer complete");

}

} catch (Exception e) {
    System.out.println("Unable to create log file.");
    System.out.println(e.getMessage());

72
public static boolean calculateTimes(long fileSize) {
    ThresholdCalculator calculator = new ThresholdCalculator();

    String uncompressedFileName = "C:/temp/NamUncompressed.txt";
    String compressedFileName = "C:/temp/NamCompressed.txt";

    boolean returnValue = false;
    try {
        double[] uncompressed = calculator.calculate(uncompressedFileName);
        double[] compressed = calculator.calculate(compressedFileName);

        int uncompressedLength = Array.getLength(uncompressed);
        int compressedLength = Array.getLength(compressed);
        double threshold = (compressed[2] - uncompressed[2])
                          / (uncompressed[0] - compressed[0]);

        double uncompressedEstimateTime = (uncompressed[1] * fileSize * fileSize)
                                           + (uncompressed[0] * fileSize) + uncompressed[2];
        double compressedEstimateTime = (compressed[1] * fileSize * fileSize)
                                        + (compressed[0] * fileSize) + compressed[2];
        returnValue = (compressedEstimateTime < uncompressedEstimateTime);
    } catch (Exception e) {
        System.out.println(e.getMessage());
        System.out.println(e.getMessage());
        System.exit(1);
    }

    return returnValue;
}
Threshold Calculator

/**
* This class is used to calculate the Threshold Value used by OPT.
* @author Alex Ridgewell
* @version 1.0
*/

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.lang.reflect.Array;
import util.StatisticsImplementor;

public class ThresholdCalculator {

/**
* The main method used to perform the calculation
* @param args
*/

public static void main(String[] args) {
    ThresholdCalculator calculator = new ThresholdCalculator();
    calculator.calculateThreshold();
}

/**
* This method calculates the Threshold Value, loading the recorded
* transmission times from file inputstreams on the file system.
* @return
*/

public String calculateThreshold() {
    String transferTime = "";
    String uncompressedFileName = "C:/temp/Uncompressed.txt";
    String compressedFileName = "C:/temp/Compressed.txt";
    String thresholdFileName = "C:/temp/threshold.txt";
    String thresholdStatisticsFileName = "C:/temp/thresholdStatistics.txt";
    try {
        FileOutputStream thresholdFile = new FileOutputStream(thresholdFileName);
        FileOutputStream thresholdStatisticsFile = new FileOutputStream(thresholdStatisticsFileName, true);
        long startTime = System.nanoTime();
        double[] uncompressed = calculate(uncompressedFileName);
        double[] compressed = calculate(compressedFileName);
        int uncompressedLength = Array.getLength(uncompressed);
        int compressedLength = Array.getLength(compressed);
        long endTime = System.nanoTime();
        thresholdFile.write(String.valueOf(threshold).getBytes());
        thresholdStatisticsFile.write(String.valueOf(threshold).getBytes());
    }
    return transferTime;
}
thresholdStatisticsFile.write(";".getBytes());
thresholdStatisticsFile.write(String.valueOf(endTime - startTime)
    .getBytes());
thresholdStatisticsFile.write(";".getBytes());
thresholdStatisticsFile.write((new java.util.Date(System
currentTimeMillis()).toString()).getBytes());
thresholdStatisticsFile.write("\r\n".getBytes());
System.out.println(threshold);

// Rename the current files and create new empty files for the new
// records.
File compressedFile = new File(compressedFileName);
compressedFile.renameTo(new File(compressedFileName
    + new java.util.Date().toString()));
compressedFile = new File(compressedFileName);
File uncompressedFile = new File(uncompressedFileName);
uncompressedFile.renameTo(new File(uncompressedFileName
    + new java.util.Date().toString()));
uncompressedFile = new File(uncompressedFileName);

transferTime = String.valueOf(endTime - startTime);
}
} catch (Exception e) {
    System.out.println(e.getMessage());
}

return transferTime;

/**
 * This method is given the path to a file containing details on earlier XML
 * transmissions and uses the StatisticsImplementor to determine the coefficients.
 * @param filename A <code>String</code> containing the path to a data file.
 * @return A <code>double[]</code> containing the coefficients
 * @throws Exception If there is a problem with the calculation.
 */
public double[] calculate(String filename) throws Exception {
    BufferedReader file = new BufferedReader(new FileReader(filename));
    String line;
    StatisticsImplementor implementor = new StatisticsImplementor();
    while ((line = file.readLine()) != null) {
        String[] dataLine = line.split(";");
        double data[] = {
            Double.parseDouble(dataLine[0].trim()),
            Double.parseDouble(dataLine[1].trim())
            + Double.parseDouble(dataLine[2].trim())
            + Double.parseDouble(dataLine[3].trim())
            + Double.parseDouble(dataLine[4].trim())
        };

        implementor.addDataPoint(data);
    }

    return implementor.setRegression();
}
Statistics Implementor

/**
 * This class provides methods to make use of the Weka statistics package.
 * It performs the linear regression calculations used to determine the Threshold Value used
 * by OPT.
 */
package util;

import weka.core.Instance;
import weka.core.Instances;
import weka.classifiers.functions.LinearRegression;
import weka.core.FastVector;
import weka.core.Attribute;

public class StatisticsImplementor {
    private Instances dataInstances;

    /**
     * Public constructor.
     */
    public StatisticsImplementor() {
        FastVector attributes = new FastVector();
        attributes.addElement(new Attribute("time"));
        attributes.addElement(new Attribute("size"));
        dataInstances = new Instances("ThresholdData", attributes, 1);
        dataInstances.setClass(new Attribute("transfer"));
        // Make the last attribute be the class
        dataInstances.setClassIndex(dataInstances.numAttributes() - 1);
    }

    /**
     * Adds a new data point to the StatisticsImplementor for future calculations.
     * @param dataPoint A double[] containing the new data point
     */
    public void addDataPoint(double[] dataPoint) {
        dataInstances.add(new Instance(1.00, dataPoint));
    }

    /**
     * Returns the coefficients determined by the linear regression
     * @return A double[] containing the coefficients
     * @throws Exception if an error occurs during the calculation
     */
    public double[] setRegression() throws Exception {
        LinearRegression regression = new LinearRegression();
        regression.buildClassifier(dataInstances);
        String method = regression.eliminateColinearAttributesTipText();
        return method;
    }
}

76
String algebra = regression.toString();
return regression.coefficients();
}
Appendix C: Sample Data

The following is a very small representative sample of the data that was recorded during the term of the research.

<table>
<thead>
<tr>
<th>File Size (Bytes)</th>
<th>Calculation Time (ns)</th>
<th>Compression Time (ns)</th>
<th>Transfer Time (ns)</th>
<th>Decompression Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>287810</td>
<td>87441</td>
<td>20843710</td>
<td>39284043</td>
<td>35094125</td>
</tr>
<tr>
<td>446710</td>
<td>134654</td>
<td>27185858</td>
<td>40077440</td>
<td>38009579</td>
</tr>
<tr>
<td>395629</td>
<td>72635</td>
<td>34554671</td>
<td>43353834</td>
<td>39658951</td>
</tr>
<tr>
<td>262965</td>
<td>88000</td>
<td>25836245</td>
<td>37637465</td>
<td>28302201</td>
</tr>
<tr>
<td>7664909</td>
<td>88000</td>
<td>429574429</td>
<td>296273968</td>
<td>194580037</td>
</tr>
<tr>
<td>703018</td>
<td>78222</td>
<td>52648870</td>
<td>66775247</td>
<td>49146470</td>
</tr>
<tr>
<td>7664909</td>
<td>88280</td>
<td>429832003</td>
<td>295769713</td>
<td>194385041</td>
</tr>
<tr>
<td>703018</td>
<td>90515</td>
<td>53006737</td>
<td>67430917</td>
<td>49451536</td>
</tr>
</tbody>
</table>