International Journal of Computer Applications (0975 – 8887)
Volume 57– No.22, November 2012

An Intelligent Protocol Algorithm to improve the Performance of Enterprise Systems Communications

Hashmi Domun
Department of Computer Science & Engineering,
University of Mauritius,
Réduit, Mauritius.

Leckraj Nagowah
Department of Computer Science & Engineering,
University of Mauritius,
Réduit, Mauritius.

ABSTRACT

Enterprise communications rely on the hardware and the network infrastructure through which clients connect to gain access to enterprise server resources and services. The communication protocols used in enterprise networks however, do not always take into account the optimization of data communications between the client and the server, and this may hamper the efficiency of the enterprise systems. The networking infrastructure that connects the clients to the server is a major source of communication inefficiency. This paper aims at proposing an intelligent protocol algorithm which dynamically senses the state of the network and determines the best mode for sending files from the server to the clients. The protocol algorithm uses multiple data compression algorithms to provide for data compression and decompression during communication. It initially learns by considering different communication scenarios whereby the protocol payload is compressed using different compression algorithms. After this learning curve, the protocol algorithm intelligently decides on and uses the best compression algorithm to optimize data transfer on the network, therefore increasing the efficiency of enterprise systems communications. Using TCP as Transport Layer protocol, the protocol algorithm can achieve up to an 80% gain in efficiency.

General Terms

Internet and Distributed Computer Systems.

Keywords

Enterprise systems, intelligent protocol algorithm, communication efficiency.

1. INTRODUCTION

Enterprise systems are important for the efficient and effective operations of the enterprise. In order for companies to enhance their business operations and become more competitive in the market, companies are realizing the benefits of implementing enterprise systems, such as Enterprise Resource Planning (ERP) systems. Indeed, ERP system implementation in the organization accounts for increased business efficiency [1]. While enterprise systems leverage the benefits of the Three-Tier architectural model [2], there are various factors that limit the performance of the enterprise systems usage, most importantly, from an end-user point of view.

Enterprise system vendors have adopted various protocols and communication schemes to maintain data integrity during communication with enterprise systems. However, such communication protocols might not efficient and the performance of enterprise systems is not heightened.

2. LITERATURE REVIEW

Enterprise systems can be described as "commercial packages that enable the integration of transactions-oriented data and business processes throughout an organization" [3]. Enterprise systems provide for "seamless integration of all the information flowing through a company—financial and accounting information, human resource information, supply chain information, and customer information" [4]. The goal of an enterprise system is to help companies streamline their business processes [5].

The client-server model is used in most enterprise systems architectural model. This model accounts for flexibility and scalability with increasing number of end-users accessing the enterprise system, since this is where "the traditional monolithic access methods begin to fail" [6].

While the appropriate hardware and network sizing enables the enterprise system to provide for intense information processing and communication, the problem often lies within the communication part. The inefficiencies of existing protocols and the transmission of uncompressed data over the network are some of the factors responsible for the increased overhead on network and computer systems.

Also, little effort has been made to understand the limitation of the existing data communication through the existing network and ways that data communication can be enhanced for the enterprise to benefit from increased efficiency and effectiveness of their enterprise systems.

Various communication protocols are used in enterprise systems to handle their communication in two scenarios, that is, between enterprise systems such as in the case of Enterprise Application Integration and between the enterprise system and the client terminal accessing the enterprise system.

There are many communication technologies that make it possible for different systems to communicate between each other. These include Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), HyperText Transfer Protocol (HTTP), Message-Oriented Middleware (MOM) and Remote Procedure Call (RPC) [6] amongst others. Simple Object Access Protocol (SOAP), Remote Method Invocation (RMI) and MQSeries are also considered [7].

In the following section, several well-established and widely adopted communication protocols and schemes, as well as emerging protocols, are analyzed with their benefits and drawbacks highlighted.

2.1 HyperText Transfer protocol (HTTP)

The HyperText Transfer Protocol (HTTP) is perhaps the most popular and widely used communication protocol on the internet, local area networks and in communication between computer systems across enterprise systems. HTTP has been designed to be a communication protocol to support communication between computers and its use for communication over the internet has been a success. The data transferred may be plain text, hypertext, images, or anything else [8].

HTTP is considered as an application protocol [9], since according to the TCP/IP model, HTTP handles the communication between an application and the underlying layer of the communication stack. Since HTTP is layered over TCP [10], the use of TCP by the HTTP protocol makes it perhaps an excellent choice because TCP employs mechanisms for data flow control and congestion prevention [11]. This accounts for the reliability of communication systems over the Internet.

The interoperability that can be achieved by using HTTP as a transport protocol [12] makes it possible for Enterprise Systems to use HTTP for enterprise communication. For instance, SOAP in the enterprise uses HTTP [13]. "Those seeking to exploit HTTP’s ubiquity — to transport SOAP over it, for instance — tend to focus on its ability to transport other protocols, an act commonly called “tunneling.” The fact that HTTP is an application protocol means that it is much more than just a transporter of bytes [12].

2.2 Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture [14] is a communication standard using the Internet InterORB Protocol (IIOP) enabling applications running on different platforms to communicate between them since CORBA provides an abstraction layer to the applications as far as the network communication is concerned [15].

However, CORBA has many drawbacks since "the disadvantages of CORBA are its complexity and variation in vendor implementation" [16] [17]. Furthermore, it has been stated [18] that old technologies like CORBA are often tied to vendor specific implementations and usually require a highly administered, costly, and complex environment to implement and manage.

Another drawback is that "many businesses and organizations have implemented service oriented style architectures using older technologies like CORBA" [18]. Enterprises will be less likely to opt for CORBA or similar technologies since these technologies will be replaced by alternatives with enhanced flexibility, scalability, performance and lesser cost to implement.

2.3 Remote Procedure Call (RPC)

RPC enables programs to call program functions across the network [19]. RPC has bindings for multiple operating systems and programming languages making it a very simple solution for cross-platform distributed system programming [20]. This means that the use of RPC in the enterprise is not limited to the platforms on which the applications are running. RPC also provides for automatic marshalling and unmarshalling of messages [19], thus reducing implementation time and easing implementation costs.

XML-RPC [21] is an extension of RPC to provide for Enterprise Information Integration "which uses XML messages traveling on HTTP to represent client-server remote procedure calls (RPC)" [22].

However, RPC also has drawbacks. It has been stated [23] that clients and servers are tightly-coupled and client applications can only invoke methods by using proprietary communication protocols. RPC is also disadvantageous whereby it does not support group communication, asynchronous communication, replication and load balancing [19].

2.4 Simple Object Access Protocol (SOAP)

SOAP is a protocol using XML messages transmitted over HTTP during request-response communications. "SOAP XML-based object serialization format can be used to perform asynchronous messaging and RPC between non-XML applications" [21], a major achievement over RPC. The XML nature of SOAP accounts for advantages like loose coupling of services and network transparency [24].

"Web services allow different applications from different sources to communicate with each other without time-consuming custom coding, and because all communication is in XML, Web services are not tied to any one operating system, programming language, or communication protocols" [23]. XML is also advantageous over the implementation of Electronic Data Interchange (EDI) for enterprise systems [25] "including readability, popularity, flexibility, heterogeneity, rich format, and low cost" [26].

However, SOAP also has various disadvantages. SOAP has been developed with "a poor set of default security features" [27]. Hence, complex and large scale implementations require additional security implementations which might be time-consuming and costly. SOAP is also responsible for "low operation speed, which is more complex problem, as developers cannot change protocol specification in order to reduce response delay" [27].

2.5 Distributed Component Object Model (DCOM)

As an extension of the Component Object Model [28], the Distributed Object Component Model [29] makes it possible to achieve software systems based on modular or component based software modules.

DCOM is considered as a high-level network protocol because it is built on top of several layers of existing protocols [30]. DCOM is an application-level protocol for object-oriented remote procedure call (ORPC). The DCOM protocol is layered on top of the OSF DCE RPC specification [31]. However, DCOM has drawbacks and limitations. Support for DCOM will most likely decrease since other technologies have quickly gained greater industry acceptance and support than predecessors like DCOM or CORBA [18]. This means that enterprises will be plagued with high maintenance cost and limitations as a result of enterprise systems evolution. Technologies like DCOM "are often tied to vendor specific implementations and usually require a highly administered, costly, and complex environment to implement and manage" [18].

2.6 SPDY

SPDY is a protocol from Google which has already been implemented in browsers such as Chromium and an Internet Draft is available [33]. The SPDY protocol has been designed "for low-latency transport of content over the World Wide Web" [34] and therefore "SPDY provide a significant improvement in speed over HTTP" [33].

SPDY is layered in such a way that "applications which run over HTTP today can work over SPDY with little or no change on behalf of the web application writer" [33]. SPDY does not entirely replace HTTP, but adopt the existing HTTP headers to maintain compatibility.

However, the design of SPDY is such that "regardless of the Accept-Encoding sent by the user-agent, the server may always send content encoded with gzip or deflate encoding" [33]. This means that the SPDY protocol compresses information prior to transmission over the network, but does not take into account whether it would be more efficient to send uncompressed data over a fast network or whether the network is slow and then in such case it would be more efficient to compress data.

3. ANALYSIS

3.1 Critical analysis of existing protocols

The analysis of existing communication protocols used in enterprises enables us to have much information on their use, their benefits but also their limitations. Some of these protocol characteristics and features are common to some protocols, if not all. These characteristics have been grouped and analyzed in the following section.

3.1.1 Heterogeneity

Protocols such as XML-RPC and CORBA are application-protocols which are specific to their scope of use and therefore these protocols are not readily open to other applications, making these protocols heterogeneous.

However, heterogeneity has been reduced to a minimum in the case of Hypertext Transport Protocol (HTTP), since HTTP is used as an application protocol which is multiplatform, open and widely adopted.

3.1.2 Stateless

HTTP is a stateless transaction-based protocol. By stateless, we mean that the server need not store any information about a client or its requests. This means that every request is completely self contained; it includes all the information needed by the server to answer the request. By transaction-based, we mean that the fundamental element of interaction is a simple transaction in which the client opens a network connection to a server, sends a single request over the connection, receives a response, and the connection is closed. This may be contrasted with session-based protocols in which connections persist over many transactions [35].

However, stateless protocols also have drawbacks. HTTP is a vulnerable, stateless protocol unsuitable for persistent state applications [36]. The problems of entity authentication, resource-access authorization, and session management are not unique to the HTTP environment [37] and information security is important in the enterprise.

3.1.3 Asynchronous

Communication protocols can be grouped into two categories, namely synchronous and asynchronous. While DCOM is mainly synchronous [38], CORBA, XML-RPC and SOAP are also synchronous, whereby the client issues a blocking request each time it invokes a service at the server side.

However it should be noted that HTTP is asynchronous. Since HTTP can be used as a wrapper for other communication protocol messages, the asynchronous nature of HTTP does not make it dependent on the communication response nor any blocking waits.

3.1.4 Flexibility

The flexibility of communication protocols is dependent on their design. While DCOM and CORBA are limited in scope to their use in the distributed computing environment, protocols like HTTP are much more flexible since it can transport any data type [8].

3.1.5 Acceptance

DCOM and CORBA are considered as ageing technologies [18] and these technologies will cause increasing costs related to support, maintenance and administration. In contrast, the nature of the HTTP protocol makes it possible to encapsulate any communication type across the network.

Moreover, the increased use of HTTP for sensitive applications has required security measures [39], for example using Secure Socket Layer (SSL) and the Transport Layer Security (TLS).

4. DESIRABLE FEATURES OF AN IDEAL PROTOCOL

An ideal protocol should be able to provide for the benefits and the desirable features amongst the analyzed features of existing protocols. The protocol we proposed therefore adopts the most desirable features of existing communication protocols.

4.1 Cross-platform compatibility

An ideal communication protocol should not have any restriction on the hardware and/or software platform on which it is being used. This is often the case in enterprises, whereby servers run on different platforms than client computers.

4.2 Transaction-based

In order to provide for ordered interactions between the client and server requests and responses, the interactions should be based on atomic or distinct transaction or events to handle the connection establishment, data request, data response and closing of the connection between the client and the server.

4.3 Flexibility of communication

An ideal communication protocol should be able to transport any kind of data on the network. The protocol should be able to act as a wrapper for other protocols and any type of data.

4.4 Data compression

Data compression enables lesser amount of information to be exchanged between computers. This will reduce the number of packets transmitted over the network and the loading time for the requested content over the network is reduced.

4.5 Stateful protocol

An ideal protocol should be able to retain useful client-server communication information so as to make decisions to make communications more efficient. This can be achieved by using an intelligent algorithm to decide on the best way of transferring information over the network.

4.6 Transparency over transport layer protocols

An ideal communication protocol should not be restricted to only a particular transport layer protocol. This means that it can use, but is not limited to, the Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Datagram Congestion Control Protocol (DCCP), Stream Control transmission Protocol (SCTP) and others.

5. DESIGN

In order to address the issues of enterprise systems communications, the solution that we proposed is to provide for an Application Layer communication protocol with real-time data compression before the data being handled by the underlying protocol, thus providing for more efficient data communication. The decision to use data compression, however, relies on the intelligent protocol algorithm to learn about all client-server communication and then decide on the best way to transfer data between the client and the server.

The ideal communication protocol should be able to transport any kind of information prior to transmission. Efficiency of information over the network is achieved through transport layer protocols such as TCP. The proposed protocol is therefore designed to represent information over the network in an optimized way so as to enhance enterprise systems communications.

5.1 High level design

A high level architecture for the proposed protocol is shown in Figure 1.

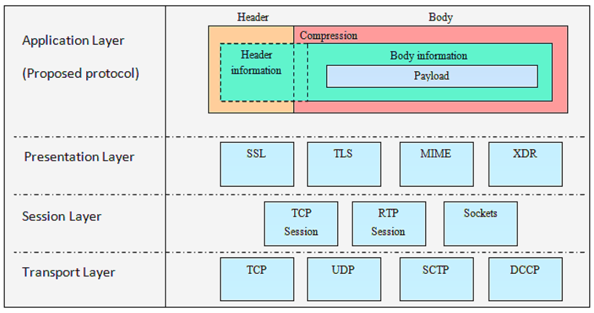


Figure 1: High-level components representation of the proposed protocol structure

To maintain compatibility with HTTP, the existing set of header messages of HTTP/1.1 are reused in the proposed protocol, while also using additional messages that extends the capabilities of the proposed protocol.

Data compression is used for the payload. This has the advantage of reducing the payload size, therefore reducing the time taken to transfer the information.

The protocol must be able to handle data compression dynamically since different data compression algorithms has different compression ratios and at different speeds.

5.2 Detailed protocol design considerations

A desirable feature of the protocol is to provide for an intelligent algorithm for adaptive data compression, since this will reduce unnecessary data compression overheads in the case of a fast network. An intelligent approach to learn and adapt to network changes means that the proposed protocol is able to decide for the best decision on the way information is exchanged between the client and the server.

In order to automatically determine whether data communication by the proposed protocol is based on a high performance, slow, congested or uncongested network, multiple factors can be considered by the protocol to take its decisions.

One way to compare a fast network to a slow network is by calculating the round trip time (RTT); that is, the time taken for a message to be sent to a destination computer and a reply sent back to the source computer. On a slow network, the RTT will be a much larger value than the RTT for a fast network. The decision to provide for data compression can thus be based on a calculated threshold value. Using RTT as a means to decide for data compression is entirely feasible since the calculated value for the RTT can be justified by a slow network, or a fast network which is heavily congested; in the latter case, the fast network will still experience increased delays in network communication.

The algorithm considers several communication scenarios during its learning curve. It initially sends a file uncompressed. When another client requests a similar file, it compresses the file using the first compression algorithm and sends the file and note the time taken and the compression and decompression time. When another client requests another similar file, another entry is added. The protocol uses a list to store information for each client-server communication. Using various metrics, the protocol algorithm takes the decision whether it is efficient to use data compression, and if so, which data compression algorithm will be most efficient for the transmission of a similar file.

5.3 Algorithm design

The protocol algorithm stores communication information in its list as follows:

```
MimeType;Filesize;RTT;Algorithm;TransferTime;Compressi
onTime;DecompressionTime
```

Using stored entries in a list, the protocol algorithm checks for similar communication scenarios for similar mime-type, file size and round trip time. The similar file size and round trip time is in a range of 0.5 to 1.5 times the current round trip time and the entry file size.

The protocol algorithm to decide on the most efficient mode of communication for each client request is described in the pseudo code below.

Step 1: Populating the list of entries and determining the communication scenarios already learned

```
Let the flag mimeExist denotes the presence of an entry with similar mime-type
Let the flag sizeExist denotes the presence of an entry with similar filesize
Let the flag rttExist denotes the presence of an entry with similar RTT
Let the list algorithmList contain all protocol entries to determine the best compression algorithm
Let file algoStat contain all protocol information
If file algoStat does not exist
        • Create empty file
        • Write header information to file
Get Mime-Type of requested resource
For each entry in file algoStat :
        • If the mime-type of the requested resource is found in the list, mimeExist = true.
        • If the requested resource filesize is within range of 0.5 to 1.5 times the protocol entries, sizeExist = true.
        • If the RTT is within range of 0.5 to 1.5 times the protocol entries, rttExist = true.
If mimeExist = true AND sizeExist = true AND rttExist = true, then add entry to list algorithmList.
For all entries in algorithmList, retrieve compression algorithm used :
If protocol entry = "none", then noneFound = true.
If protocol entry = "GZip", then gzipFound = true.
If protocol entry = "ZIP", then zipFound = true.
If protocol entry = "Bzip2", then bzip2Found = true.
```

Step 2: Deciding on the most efficient communication mode

```
If noneFound = true AND gzipFound = true AND zipFound = true AND bzip2Found = true, then
    • This means that similar scenarios have already been taken into account by the protocol. Call method to return the best algorithm from the list of protocol communication entries by returning the entry having the least time to compress, transfer and decompress data.
Else
    • This means that at least one communication scenario has not been tested. Call method to return a possible algorithm that can be considered which is not present in algorithmList.
End If
```

5.4 Low level design

The proposed protocol comprises of a Header and a Body section, as shown in Figure 2:

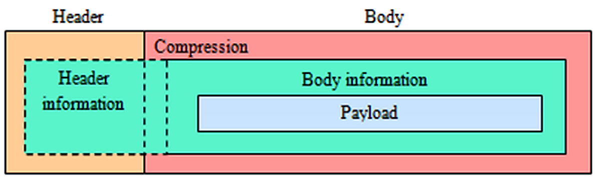


Figure 2: Proposed protocol header and body

Compatibility with the HTTP protocol is maintained by adopting the HTTP header messages as defined in RFC 2616. The proposed protocol also uses additional request-response messages which are useful especially in an Enterprise context.

Figure 3 shows the interactions between the client and server components.

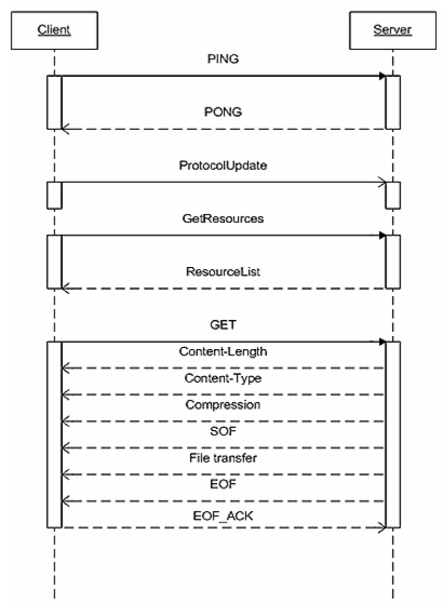


Figure 3: Interactions between the client and the server

The implementation scope of our proposed protocol focuses on the intelligent use of data compression to determine the most efficient data compression algorithm according to the state of the network. The algorithm is able to learn from the different compression algorithms and becomes increasingly efficient over time.

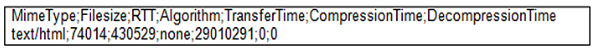
The proposed protocol has been implemented in Java using available libraries for data compression and decompression. The three compression algorithms used for the implementation are Zip, Gzip and Bzip2.

The protocol algorithm first learns about the different communication scenarios with and without data compression and then takes decisions as to which communication mode is most efficient. This means that the protocol algorithm does have a learning curve. However the greater the number of client requests, the faster the protocol learns.

6. TESTING

In order to weigh the efficiency of the proposed protocol, different test scenarios have been devised to analyze the effect of having an intelligent algorithm increasing the efficiency of network communication between a client and a server. A test scenario has been devised whereby two clients programs, Client 1 and Client 2, each request a HTML file and a JPEG file respectively from the server.

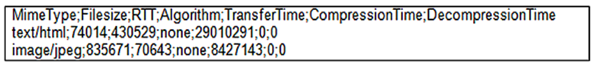
When Client 1 requests the HTML file, the protocol stores the communication information in the list after receiving the acknowledgement as shown in Figure 4.



```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
```

Figure 4: Protocol entries for Client 1 HTML request

When Client2 requests the JPEG file for the first time, another entry is stored by the protocol as shown in Figure 5.

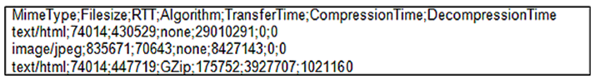


```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
image/jpeg;835671;70643;none;8427143;0;0
```

Figure 5: Protocol entries for Client 2 JPEG request

For each request that the server receives, the algorithm checks if a communication scenario has been saved by the protocol. For the first time, the protocol considers no compression.

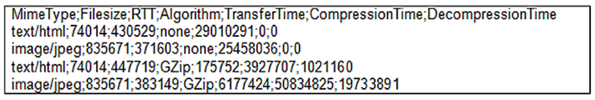
For a second file transfer, the protocol considers Gzip compression algorithm as a possible scenario and stores an entry after the acknowledgement as shown in Figure 6.



```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
image/jpeg;835671;70643;none;8427143;0;0
text/html;74014;447719;GZip;175752;3927707;1021160
```

Figure 6: Protocol entries with new Client 1 entry

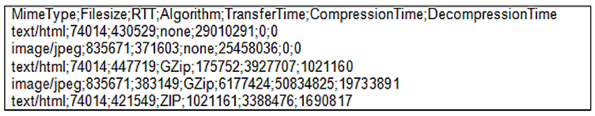
The Gzip compression is also considered for the second JPEG data transfer for Client 2 as shown in Figure 7.



```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
image/jpeg;835671;371603;none;25458036;0;0
text/html;74014;447719;GZip;175752;3927707;1021160
image/jpeg;835671;383149;GZip;6177424;50834825;19733891
```

Figure 7: Protocol entries with new Client 2 request

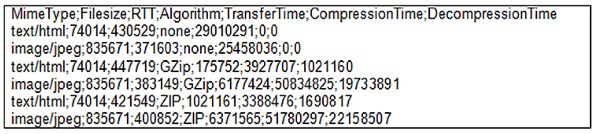
For the third file transfer for Client 1, the protocol considers the Zip compression algorithm as a possible scenario and updates its list as shown in Figure 8.



```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
image/jpeg;835671;371603;none;25458036;0;0
text/html;74014;447719;GZip;175752;3927707;1021160
image/jpeg;835671;383149;GZip;6177424;50834825;19733891
text/html;74014;421549;ZIP;1021161;3388476;1690817
```

Figure 8: Third file transfer for Client 1

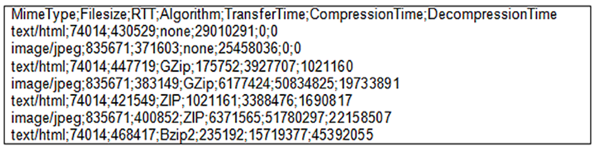
Similarly, the Zip algorithm is used and the list updated as shown in Figure 9.



```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
image/jpeg;835671;371603;none;25458036;0;0
text/html;74014;447719;GZip;175752;3927707;1021160
image/jpeg;835671;383149;GZip;6177424;50834825;19733891
text/html;74014;421549;ZIP;1021161;3388476;1690817
image/jpeg;835671;400852;ZIP;6371565;51780297;22158507
```

Figure 9: Third file transfer for Client 2

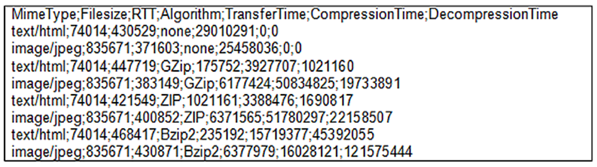
For the fourth file transfer for Client 1, the protocol uses the Bzip2 compression algorithm as a possible scenario and updates its list after the data transfer as shown in Figure 10.



```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
image/jpeg;835671;371603;none;25458036;0;0
text/html;74014;447719;GZip;175752;3927707;1021160
image/jpeg;835671;383149;GZip;6177424;50834825;19733891
text/html;74014;421549;ZIP;1021161;3388476;1690817
image/jpeg;835671;400852;ZIP;6371565;51780297;22158507
text/html;74014;468417;Bzip2;235192;15719377;45392055
```

Figure 10: Fourth file transfer for Client 1

Similarly, the protocol uses the Bzip2 compression algorithm Client 2 and updates its list as shown in Figure 11.



```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
image/jpeg;835671;371603;none;25458036;0;0
text/html;74014;447719;GZip;175752;3927707;1021160
image/jpeg;835671;383149;GZip;6177424;50834825;19733891
text/html;74014;421549;ZIP;1021161;3388476;1690817
image/jpeg;835671;400852;ZIP;6371565;51780297;22158507
text/html;74014;468417;Bzip2;235192;15719377;45392055
image/jpeg;835671;430871;Bzip2;6377979;16028121;121575444
```

Figure 11: Fourth file transfer for Client 2

For the fifth file transfer for Client 1 and Client2, the protocol decides on the best algorithm since all possible scenarios have been learned.

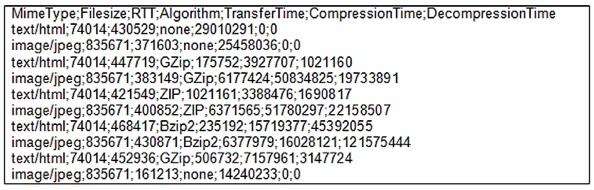
Table 1. Protocol communication entries

Mime Type	File size (KB)	RTT (ms)	Algo	Transfer (ms)	Compr ession (ms)	Decomp ression (ms)
text/ html	74	430	none	29010	0	0
image/ jpeg	836	371	none	25458	0	0
text/ html	74	447	GZip	175	3927	1021
image/ jpeg	836	383	GZip	6177	50834	19733
text/ html	74	421	ZIP	1021	3388	1690
image/ jpeg	836	400	ZIP	6371	51780	22158
text/ html	74	468	Bzip2	235	15719	45392
image/ jpeg	836	430	Bzip2	6377	16028	121575

Table 2. Compression algorithm comparison table

Compression Algorithm	Client 1: Total time taken (ms)	Client 2: Total time taken (ms)
none	29010	25458
Gzip	5124	76746
Zip	6100	80310
BZip2	61346	143981

From the results obtained as shown in Table 1, the protocol compares the total time taken for each file request as shown in Table 2. The protocol therefore chooses Gzip as the best algorithm for Client 1 (HTML file) while no compression algorithm is considered for Client 2 (JPEG file). After deciding on the most efficient communication mode for each client, the protocol updates its communication entries after receiving acknowledgements from each client as shown in Figure 12.



```
MimeType;Filesize;RTT;Algorithm;TransferTime;CompressionTime;DecompressionTime
text/html;74014;430529;none;29010291;0;0
image/jpeg;835671;371603;none;25458036;0;0
text/html;74014;447719;GZip;175752;3927707;1021160
image/jpeg;835671;383149;GZip;6177424;50834825;19733891
text/html;74014;421549;ZIP;1021161;3388476;1690817
image/jpeg;835671;400852;ZIP;6371565;51780297;22158507
text/html;74014;468417;Bzip2;235192;15719377;45392055
image/jpeg;835671;430871;Bzip2;6377979;16028121;121575444
text/html;74014;452936;GZip;506732;7157961;3147724
image/jpeg;835671;161213;none;14240233;0;0
```

Figure 12: Protocol entries with the most efficient communication for each data type

7. DISCUSSION AND EVALUATION

The primary aim of this research project is to investigate the causes of inefficient data communication in enterprise systems and to propose a new way to enhance the communication performance in enterprise systems. From the results and findings, it is clear that inefficient communication has a negative effect on enterprise communication performance.

The intelligent protocol that we developed with a server and client application to transfer different data types is able to largely enhance the communication between the client and the server. The protocol algorithm successfully learns from all client-server communication which allows it to consider the most efficient means of communication for each data transfer. The algorithm is also able to adapt to new data types and becomes most efficient at the end of its learning curve.

Three compression algorithms have been used in this protocol implementation, namely Gzip, Zip and Bzip2. The data compression functionality used by the protocol can also be extended to other compression algorithms as well. While SPDY offers, by design, data compression through Gzip or Deflate algorithms by default, the proposed protocol offers the choice of either compressing the data or not, and if so, which compression algorithm would be best.

An important point to consider is that the protocol first builds up a list of communication scenarios before deciding on the best algorithm. Therefore the most efficient communication is determined after a number of learning trials. As the protocol handles more and more client requests, the protocol algorithm is able to learn faster by building up the communication entries faster but also allowing the protocol to perform data analysis and comparison with a larger pool of data.

The effectiveness of data compression, however, varies for different file types. For instance, a HTML file of size 74KB can be compressed to 13.6KB using Zip algorithm. The transfer time is therefore much less than for uncompressed data transfer, even when taking into account the compression and decompression time. On the other hand, a 836KB JPEG image can be compressed to about 799KB only. Also, this implementation takes into account the data compression time and decompression time as well, if any. If the compression and decompression time is not taken into account, then only the communication with least transfer time is considered.

The efficiency achieved with our proposed protocol for different data types has been calculated and listed in Table 3.

Table 3. Efficiency comparison for different scenarios

File	**Worst case (ms)**	**Best case (ms)**	**Gain (ms)**	**% Gain**
HTML	73.47	8.04	65.43	89.06
CSS	2.43	0.76	1.67	68.64
HTML+CSS +JS+JPEG	2150.40	765.95	1384.45	64.38
HTML	29.01	5.12	23.89	82.34
HTML+ JPEG	544.68	305.82	238.85	43.85
Average efficiency				69.7

From the communication scenarios listed above, our proposed protocol can achieve an average of 69.7% communication efficiency increase over a range of data types while the efficiency increase for HTML files is more than 80%.

A fast communication will have the enterprise tasks done quickly and effectively while inefficient communication will not be able to heighten the efficiency and use of the enterprise system [40]. Slow communication also means that users will be more frustrated with their tasks taking time to complete and not being able to be productive [41].

8. CONCLUSION AND FUTURE WORKS

This section highlights the achievements of this research project, along with limitations encountered and future works to further enhance the protocol.

In this research paper, the protocol algorithm that we proposed highlights the efficiency increase in communication that can be achieved and therefore heighten enterprise system communications performance. The algorithm can achieve an average of 69.7% communication efficiency increase over a range of data types while the efficiency increase for HTML files is more than 80%. The protocol uses an intelligent algorithm that learns through all communications and is able to decide on the most efficient means of communication. The algorithm can also adapt to new data types and is most efficient after its learning time.

The protocol is not only suitable for the enterprise, but also for any computer system, since the design does not restrict the protocol only in the enterprise context. By addressing issues regarding enterprise systems communications, the enterprise is able to operate more efficiently and effectively. Without having a proper means to enhance enterprise communication performance means that the enterprise system will not be used efficiently.

Although the protocol algorithm brings a considerable increase in communication efficiency, it can be further improved in the following ways. E.g. the sorting algorithms can be enhanced by using more efficient algorithms. The comparison of the round trip time and the file size can be enhanced so as for the comparison thresholds to vary dynamically. Also, the protocol can be made to use a wider range of compression algorithms. The protocol algorithm can be made to estimate the comparison values to reducing the computational time before sending the data over the network. The CPU usage can also be taken into account by the protocol algorithm during data compression and decompression so as to consider the battery life of mobile devices. Last but not least, the protocol headers can also be compressed prior to data transfer over the network. This will help to achieve more than the efficiency increase that our proposed protocol can achieve.

9. REFERENCES

[1] Markus, M. and Tanis, C. 1999, "The Enterprise Systems Experience – From Adoption to Success", in Framing the Domains of IT Research: Glimpsing the Future Through the Past, Pinnaflex Educational Resources Inc., Cincinnati, pp. 1-46.

[2] Gupta, A. 2000, "Enterprise resource planning: the emerging organizational value systems", Industrial Management & Data Systems 2000; 103(3): pp. 114-18.

[3] Markus, M. and Tanis, C. 2000, "The enterprise system experience - from adoption to Success", in Zmud, R.W. (Ed.), "Framing the Domains of IT Management: Projecting the Future Through the Past", Pinnaflex Educational Resources, Inc., Cincinnatti, OH, pp. 173-207.

[4] Davenport, T. 1998, "Putting the Enterprise into the Enterprise System", Harvard Business Review, July-August, pp. 121–131.

[5] Fan, M., Stallaert, J. & Whinston, A. 2000, "The adoption and design methodologies of component based enterprise systems", European Journal of Information Systems, pp. 25-35.

[6] Rezayat, M. 1999, "The Enterprise-Web portal for life-cycle support", Computer-Aided Design 2000, 32(2), pp. 85–96.

[7] Fremantle, P., Weerawarana, S. and Khalaf, R. 2002, "Enterprise services", Communications of the ACM 45(10), pp. 77-82

[8] Berners-Lee, T., Caillau, R., Luotonen, A., Nielsen, H.F. & Secret, A. 1994, "The world-wide web". Communications of the ACM 37(8), pp. 76–82.

[9] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. 1999, "Hypertext Transfer Protocol — HTTP/1.1," Internet Engineering Task Force RFC 2616, June 1999. [Online]. Available at <http://www.ietf.org/rfc/rfc2616.txt>

[10] Heidemann, J., Obraczka, K. and Touch, J. 1997, "Modeling the Performance of HTTP Over Several Transport Protocols", IEEE/ACM Transactions on Networking, 5(5), 1997, pp. 616–630.

[11] Jacobson, V. 1988, "Congestion avoidance and control", Proceedings of the ACM SIGCOMM'88, pp. 314-329.

[12] Vinoski, S. 2002, "Web services Interaction Models, Part 2", IEEE Internet Computing, vol. 6, no 3 (2002), pp. 89–91.

[13] Leong, K. S, Ng, M. L. & Engels, D. W. 2004, "EPC network architecture", Auto-ID Labs Research Workshop, 2004. [Online]. Available at: <http://www.m-lab.ch/auto-id/SwissReWorkshop/agenda.html>

[14] CORBA 1998, Common Object Services Specification, available at <ftp://ftp.omg.org/pub/docs/formal/98-07-05.pdf>

[15] Gritzalis, S., Iliadis, J. and Oikonomopoulos, S. 2000, "Distributed component software security issues on deploying a secure electronic marketplace", Information Management & Computer Security, Vol. 8 Iss: 1, pp.5-13

[16] Frankewitsch, T. and Prokosch, H. 2000, "Graphical navigation of the UMLS metathesaurus on a locally installed database implementing CORBAmed's Lexicon Query Service", Proceedings of the AMIA Symphony, 2000, pp. 1009

[17] Muller, M., Ganslandt, T., Eich, H. P., Lang, K., Ohmann, C. & Prokosch, H. 2001, "Towards integration of clinical decision support in commercial hospital information system using distributed, reusable software and knowledge components", International Journal of Medical Informatics 64, pp. 369-377.

[18] Henzel, J., Hutchinson, B. & Thwaits, A. 2006, "Using web services to promote library-extension collaboration", Library Hi Tech, Vol. 24 Issue: 1 pp. 126 – 141.

[19] Pinus, H. 2004, "Middleware: Past and present a comparison", [Online]. Available at <http://userpages.umbc.edu/~dgorin1/451/middleware/middleware.pdf>

[20] Nunn, R. 2003, "Distributed software architectures using middleware", available at <http://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/3C05-02-03/aswe18-essay.pdf>

[21] Pokorný, J. 2009, "Xml in enterprise systems", Informatica 20, pp. 417–438.

[22] Harold, E. R. & Means, W. S. 2002, "XML in a Nutshell", O'Reilly.

[23] Wei, H. and Godfrey, T. 2008, "Database Middleware and Web Services for Data Distribution and Integration in Distributed Heterogeneous Database Systems",

[24] Pulier, E. and Taylor, H. 2006, "Understanding Enterprise SOA", Manning Publication Co., Greenwich.

[25] Huang, S., Kwan, I., Yen, D. & Hsueh, S. "Developing an XML gateway for business-to-business commerce", Proceedings of the First International Conference on Web Information Systems Engineering, Hong Kong, China, June 19–20, 2000, pp. 67 –74.

[26] Tao, Y.-H., Hong, T.-P., & Sun, S.-I. 2004, "An XML implementation process model for enterprise applications", Computers in Industry, Vol. 55: 181-196.

[27] Doroshenko, A. and Yatsenko, K. 2007, "Protocols for Mobile Devices Integration in Heterogeneous Environments", Information systems technology and its applications, 6th international conference ISTA`2007 May 23-25, 2007, Kharkiv, Ukraine.

[28] Microsoft Corporation and Digital Equipment Corp. 1995, "The Component Object Model Specification", Oct. 1995. [Online]. Available at <http://www.microsoft.com/oledev/olecom/title.htm>

[29] Brown, N. and Kindel, C. 1996, "Distributed Component Object Model Protocol -- DCOM/1.0", Internet Draft, [Online]. Available at <http://www.microsoft.com/oledev/olecom/draft-brown-dcom-v1-spec-01.txt>

[30] Eddon, G. and Eddon, E. 1998, "Understanding the DCOM Wire Protocol by Analyzing Network Data Packets", Microsoft Systems Journal, Microsoft Corporation, March 1998. [Online]. Available at <http://www.microsoft.com/msj/0398/dcom.aspx>

[31] OSF DCE RPC Specification, The Open Group, 1994. [Online]. Available at <http://www5.opengroup.org/dce/>

[32] Wang, Y., Damani, O. and Lee, W. 1997, "Reliability and Availability Issues in Distributed Component Ojbect Model (DCOM)", Fourth International Workshop on Community Networking Proceedings, 1997, pp. 59 –63.

[33] Belshe, M. and Peon, R. 2012, "SPDY Protocol", Internet Draft, Network Working Group, Internet Engineering Task Force, Feb 2012. Available at <http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00>.

[34] Rosenberg, S., Dangi, S. and Warnakulasooriya, I. 2012, "Data and Network Optimization Effect on Web Performance", Sillicon Valley Campus, Carnegie Mellon University, Mountain View, CA 94035 [Online]. Available at <http://repository.cmu.edu/cgi/viewcontent.cgi?article=1091&context=silicon_valley>

[35] Farquhar, A., Fikes, R., Pratt, W. and Rice, J. 1995, "Collaborative ontology construction for information integration". Technical Report KSL-95-63, Stanford University Knowledge Systems Laboratory, 1995.

[36] Zanero, S., Carettoni, L. and Zanchetta, M. 2005, "Automatic Detection of Web Application Security Flaws", Black Hat Briefings, 2005.

[37] Gutzmann, K. 2001, "Access control and session management in the HTTP environment". IEEE Internet Computing, January/February 2001.

[38] Adamopoulos, D. X., Pavlou, G. and Papandreou, C. A. 1999, "Distributed Object Platforms in Telecommunications: A Comparison Between DCOM and CORBA". British Telecom. Eng. 18 pp. 43-49.

[39] Rescorla, E. 2000, "HTTP Over TLS", Internet Engineering Task Force (IETF).

[40] Guynes, J. 1988, "Impact of System Response Time on State Anxiety", Communications of the ACM, vol. 31, no. 3, 1988, pp. 342-347.

[41] Tolia, N., Andersen, D. & Satyanarayanan, M. 2006, "Quantifying Interactive User Experience on Thin Clients", Computer Science Department, Carnegie Mellon University, Paper 7.