Evaluating SOAP for High Performance Applications in Capital Markets

Christopher Kohlhoff
Tenermerx Pty Ltd
6/50 Ben Boyd Rd, Neutral Bay
NSW 2089 Australia
chris@tenermerx.com

Robert Steele
University of Technology, Sydney
PO Box 123 Broadway
NSW 2007 Australia
rsteele@it.uts.edu.au

1 August 2004

Abstract

Web services, with an emphasis on open standards and flexibility, may provide benefits over existing capital markets integration practices. However, web services must first meet certain technical requirements including performance, security and fault–tolerance. This paper presents an experimental evaluation of SOAP performance using realistic business application message content. To get some indication of whether SOAP is appropriate for high performance capital markets systems, the results are compared with a widely used existing protocol. The study finds that, although SOAP performs relatively poorly, the difference is less than in scientific computing environments. Furthermore, we find that in realistic business applications it is possible for text–based wire formats to have comparable performance to binary, and that the

text–based nature of XML is not sufficient to explain SOAP's inefficiency. This suggests that further work may enable SOAP to become a viable wire format for high performance business applications.

1 Introduction

Over the last decade, rapid advances in computing technology have driven dramatic changes in the financial sector. The provision of online services like real–time share trading has prompted a move to new business models such as global markets, 24–hour trading, and straight–through processing [23]. Tremendous growth in the number of third party financial networks has increased competition and accelerated the race to develop advanced, automated trading systems [8].

To compete effectively, existing organisations need to form alliances and provide integrated services. However, business–to–business integration is not a new concept for capital markets. The financial domain has utilised industry standard protocols for the integration of distributed applications since the 1970s [28].

Web services are now emerging as a technology for systematic and flexible application–to–application integration [6]. Web services differ from most existing integration practices in that they utilise established, proven web protocols and open XML standards.

Capital markets systems may benefit from the introduction of web services. Existing integration practice is characterised by competing industry standards and many proprietary protocols, but web services' emphasis on open standards may be an advantage in getting industry players to agree. The use of XML, and its formal definition language XML Schema, can help improve integration protocol implementations by eliminating ambiguities, as well as providing support for automatic validation of messages. Finally, the extensibility of web services and XML can allow integration mechanisms to evolve as markets require new functionality, without causing further fragmentation

of protocols.

However, before web services can be used for capital markets systems, various technical requirements must be met. These requirements include performance, security and fault–tolerance.

This study examines the performance of SOAP in realistic business computing scenarios. More specifically, the primary goal of this research is to consider the feasibility of using SOAP in capital markets systems, and particularly in real–time trading systems. Unlike our previous study [18], which used randomly generated values to simulate realistic data, the research in this paper uses real data supplied by the Australian Stock Exchange.

The approach adopted by this study is to evaluate the performance of SOAP against existing practice. The study compares the performance of SOAP with the established, widely used, domain–specific protocol, FIX. The relative performance of SOAP and FIX may be useful in determining whether SOAP can meet the performance requirements of capital markets.

The emphasis of this study was on the inherent performance limitations of the wire formats. Both SOAP and FIX use a text–based wire representation, and therefore it may seem reasonable to conclude, based on the existing research, that both would be impacted by the same inherent inefficiencies. For this reason, a binary wire format, CDR, was included in the comparison to gauge the costs associated with text encoding.

The study finds, firstly, that in business applications SOAP does indeed perform poorly compared to the binary wire format, CDR. SOAP messages are some 2–3.5 times the size of the equivalent CDR messages. Latency over local networks is substantially increased, with encoding and decoding 4–8 times more expensive. These results are similar to the conclusions of earlier studies, although the results show a less marked difference than when the focus is on transmission of numerical data.

When compared to FIX, SOAP again exhibits poorer performance. SOAP messages are 4–5.5

times the size of FIX, latency is 2–3 times worse, and encoding/decoding costs are increased by up to 8 times.

Given that FIX, like SOAP, is text–based, the surprising result is that FIX performed comparably to CDR. From this we have been led to conclude that, in realistic business application scenarios, SOAP's poor performance cannot be adequately explained simply by the disadvantages of text–based over binary wire formats. This also suggests that improvements in the efficiency of SOAP encoders and decoders may improve its suitability for use in high performance business applications.

A final result of this study is that SOAP can be used as it is in certain small–to–medium real–time trading applications. Using the actual trading data supplied by the Australian Stock Exchange, the research findings suggest that SOAP can adequately cover the peak throughput requirements for that system. The caveat is that scaling to larger numbers of clients will have proportionally greater hardware and network bandwidth needs than either FIX or CDR.

2 Background

Software systems used in capital markets can be classified according to their position in the trading lifecycle [23]:

- **Pre–trade services.** The delivery of real–time and historical market data, analysis of this data, and routing of an order to the best trading entity for a transaction.
- **Trading.** The execution of a trade itself at a trading entity, such as an exchange like the Australian Stock Exchange (ASX).
- **Post–trade services.** Operations performed prior to finalisation of a trade, such as surveillance, compliance checking, or risk management.

- **Settlement.** Finalisation of a trade by transferring money between the buyer and seller.
- **Registry.** Transfer of ownership of the securities from seller to buyer.

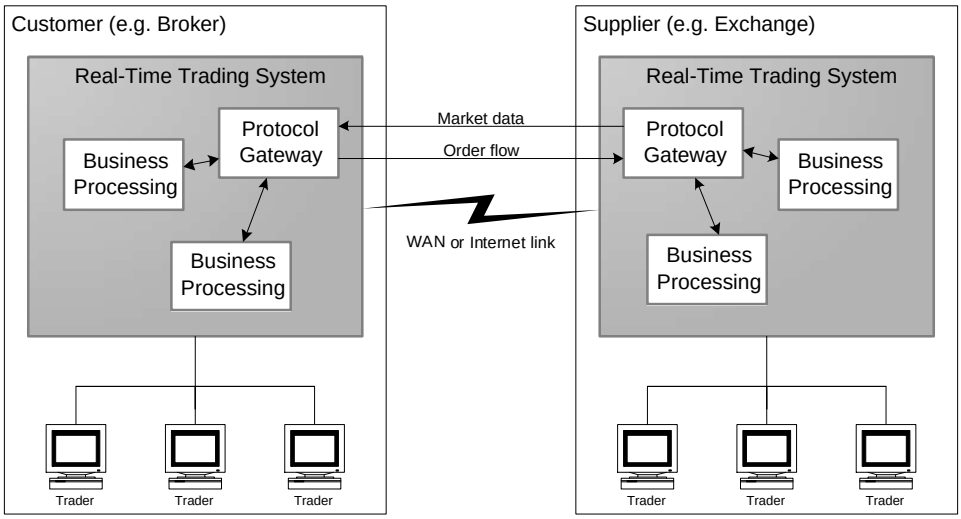

Figure 1: Integration between real–time trading systems

This paper will focus on the pre–trade part of the lifecycle, and in particular the integration needs of real–time trading systems. Integration between real–time trading systems typically involves the communication of live market data as well as the flow of buy and sell orders, as shown in Figure 1. Given the potentially large volumes of data and the need for timely delivery, integration between real–time trading systems has, in the authors' experience, the highest performance requirements in the domain.

2.1 FIX Protocol

The Financial Information eXchange (FIX) protocol [11] is a messaging standard developed specifically for the real–time electronic exchange of securities transactions.

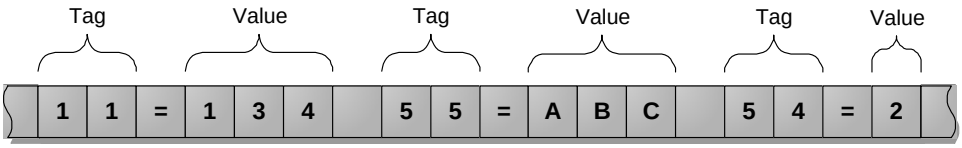

Figure 2: An example FIX protocol message fragment

FIX messages are text–based, and consist of tag–value pairs separated by a special delimiter character (SOH, which is ASCII value 0x01) as illustrated by Figure 2. The tags are short strings of digits, and types of values include strings, integers, floating point values, timestamps and arbitrary binary data. Although the content of a message is represented by complex application structures, the layout of an encoded message is flat with flexible ordering of fields. The protocol specification describes, in natural language, the set of available tags, their corresponding business meanings, and the required message structure.

Recent versions of the FIX protocol have introduced an XML–based message format, called FIXML [10]. This provides FIX messages with a rich on–the–wire structure, enabling automated validation and reducing the inherent ambiguities of the tag–based approach. XML also allows the FIX standard to evolve to include new functionality without causing further version fragmentation.

As this paper evaluates the suitability of SOAP for capital markets systems, the FIX protocol will be used as the basis for some comparisons. FIX has been selected for this purpose over other industry protocols due to its wide usage. A 1999 survey of market participants, referenced in [13], found that 82% of surveyed brokers used FIX. The influence of FIX also extends to many organisations that use variants of the standard protocol, or use protocol message definitions that may be classed as FIX–like, such as the ASX's SEATS Open Interface [1].

3 Related Work

Several studies have evaluated the performance of SOAP and XML [7, 14, 4]. These studies all agreed that SOAP and XML incur a substantial performance penalty compared to binary protocols.

[7] conducted an experimental evaluation of the latency performance of various SOAP implementations, comparing with other protocols such as Java RMI and CORBA/IIOP. A conclusion drawn from these results was that SOAP is orders of magnitude slower, although for some of the slowest SOAP systems this can be partly explained by poor implementation.

[14] evaluated the performance of SOAP for high performance scientific computing. Their experiments compared Java RMI with SOAP by sending large arrays of doubles (i.e. floating point values with 18 decimal digits of precision). The results showed that SOAP is much slower than Java RMI, typically by about a factor of ten. They concluded that SOAP's XML messages were inherently unsuitable for use in transferring bulk data, but due to the format's flexibility and accessibility, may be useful as part of a multi–protocol system with SOAP as a "*lingua franca*".

[4] presented the results of experiments that compared the encoding, decoding and network performance of various message formats, including XML. They found that the marshalling and communications costs of XML are staggeringly high in comparison to more traditional approaches, with XML some *2 to 4 orders of magnitude* slower in encoding and decoding than CORBA/IIOP and similar binary wire formats. They concluded that XML wire formats are inappropriate for high performance systems, as the baseline performance of all systems is strongly determined by their wire format.

These studies identified some factors that can affect the performance of web services and SOAP, which can be broadly grouped into three main categories.

3.1 Quality of Implementation

Design and implementation decisions made by SOAP infrastructure vendors can have a considerable impact of performance. These factors include:

- **Choice of XML parsing method.** Different XML parsing models have different trade–offs with respect to memory efficiency, computational speed, and ease of use. Of the generic XML parsing models, pull parsing offers the highest performance, as well as high memory efficiency [15]. However, [5] found that schema–specific parsers can greatly enhance performance compared to general purpose XML parsers, particularly where large data structures are involved.

- **Message length calculation.** When using SOAP with an HTTP/1.0 network binding, the length of the body must be specified in the "Content–Length" header field [3]. Setting the HTTP Content–Length field is difficult for dynamic data, because the message must be buffered as it is constructed to determine how long it is, and the message itself is not sent until encoding is finished [7].

- **Connection establishment costs.** HTTP/1.0 [3] mandates the establishment of a new connection for each operation. Establishing a new connection for each transaction can have a negative impact on performance due to interaction with certain TCP features [25], such as the three–way handshake [26] and slow start algorithm [17]. When looking at HTTP/1.0 performance over the Internet, at least one quarter of the transaction time may be taken up by connection establishment [19]. HTTP/1.1 provides a connection keep–alive feature that allows a client to perform multiple operations over a single connection [9].

- **Pipelining** HTTP/1.1 adds support for *pipelining* [9], which is the ability to send multiple requests on the connection before waiting for a response. This allows the connection to

be used more efficiently. As an alternative to pipelining, an HTTP/1.0 implementation can make multiple single–operation connections in parallel, typically by using multithreading. [21] found that an HTTP/1.1 implementation using buffered pipelining will use less than 10% of the number of TCP packets that HTTP/1.0 does, and execute in less elapsed time. However HTTP/1.1 without pipelining has a higher elapsed time than HTTP/1.0 using multiple connections. This indicates that connection keep–alive alone is insufficient to improve performance.

- **Inappropriate TCP options.** [7] found that some SOAP implementations suffered from significantly worse performance due to interaction between the Nagle algorithm and the TCP delayed acknowledgement algorithm in the operating systems for the client and server [26]. Clearly this is not an inherent problem with either SOAP or HTTP.

3.2 Network Protocol Binding

The FIX protocol defines a session as a "bi–directional stream of ordered messages between two parties" [11], and so there are no request–response semantics imposed by its specification. Consequently, when seeking to apply SOAP to a real–time trading system we would prefer to use messaging–style rather than RPC–style communication. Since HTTP is a request–response protocol [9] with strict client and server roles, it may be ill–suited to use in message–style communication.

Fortunately, SOAP does not specify a particular network transport binding, and using SOAP with alternative network protocols may offer performance advantages. This clearly implies that the inefficiencies attributed to HTTP are not inherent to SOAP.

3.3 Inherent Limitations of SOAP

Open metadata technologies such as XML can provide a large gain in usability, but the success of these technologies requires that their use does not unreasonably degrade performance [30].

XML is extremely robust with respect to changes in the format of the incoming record [4]. However, the use of XML can negatively impact the performance of SOAP in the following areas:

- **Speed of encoding and decoding.** The conversion of data from binary to ASCII and vice–versa is the major performance cost of XML [4]. This is particularly the case for floating point values, which were found constitute the major cost of XML encoding and decoding [5]. The use of a text–based protocol also precludes the application of optimisations available to binary protocols when communication occurs between homogeneous systems [4].

- **Message size.** For XML, an expansion factor of 6–8 times over the original binary data is not unusual [4]. [5] measured expansion at 4 to 10 times, and [14] found that SOAP's data representation size is typically about 10 times the size of the equivalent binary representation. This substantially greater size may result in higher network transmission costs and increased latency.

One suggested strategy for overcoming these inherent performance inefficiencies is the use of binary XML representations [30, 12, 27, 20].

4 Experimental Design

The FIX protocol, like XML and SOAP, is text–based [11]. This means that FIX has the same performance issues with regard to the encoding and decoding of numerical data. Similarly, FIX

messages may be larger than their equivalent binary representation, although overhead is lower than for XML due to FIX's compact tag–value format.

The focus of this study is on the inherent performance issues of the SOAP and FIX wire formats. With this in mind, the experiments were designed to eliminate quality of implementation and network protocol factors from consideration. This was done by sending messages encoded in the various wire formats over "raw" TCP sockets, using a consistent network programming model in each case. SOAP bindings such as HTTP were not used. Furthermore, initial transmissions were excluded from the results to eliminate effects from the TCP slow start algorithm [17], and the TCP_NODELAY option was turned on to disable the Nagle algorithm [26].

To aid in the identification of performance issues associated with text–based wire formats, comparisons were also made with a binary wire format. The Common Data Representation (CDR) [22], which is used as the basis of CORBA communication, was selected for this purpose.

Three types of experiment were conducted:

- **Message Size.** The encoded representation of the application structure in the three wire formats was compared.

- **Latency.** Test programs were written to measure round trip times for sending a single message using each of the wire formats. High resolution timers enabled the separation of the times spent on encoding, decoding and network transmission.

- **Throughput.** These tests attempted to measure the amount of time required to transmit, in a single direction, a large volume of messages. The first set of tests measured throughput from application data structure at one end to application data structure at the other. Comparing with the throughput of pre–encoded messages allows a determination as to whether encoding/decoding costs or the network was the limiting factor.

4.1 Data

```
msg.header.SenderCompID = “ABC”
msg.header.TargetCompID = “XYZ”
msg.header.MsgSeqNum = 1234
msg.header.SendingTime = “20021128-18:09:42”

msg.body.MarketDataInc.MDEntry[0].MDUpdateAction = New
msg.body.MarketDataInc.MDEntry[0].MDEntryType = Trade
msg.body.MarketDataInc.MDEntry[0].MDEntryID = “T068224”
msg.body.MarketDataInc.MDEntry[0].Instrument.Symbol = “CBG”
msg.body.MarketDataInc.MDEntry[0].Instrument.CFICode = “ESXXXX”
msg.body.MarketDataInc.MDEntry[0].MDEntryPx = 6.35
msg.body.MarketDataInc.MDEntry[0].MDEntrySize = 451
msg.body.MarketDataInc.MDEntry[0].MDEntryDate = “20021125”
msg.body.MarketDataInc.MDEntry[0].MDEntryTime = “10:18:58”
msg.body.MarketDataInc.MDEntry[0].SellerDays = 3
msg.body.MarketDataInc.MDEntry[0].MDEntryBuyer = 2402
msg.body.MarketDataInc.MDEntry[0].MDEntrySeller = 3713
. . .
```

Figure 3: An example application data structure for the market data incremental refresh message carrying a trade report

The “market data incremental refresh” FIX Message was selected as the business data for the experiment. This message is used for sending market data updates, such as the latest stock prices or reports of new trades, throughout a trading day and would typically be high volume and time critical.

The test messages were generated from the ASX’s Signal C wire format [2] data for a single day. Signal C is an electronic signal that provides subscribers with trades and quotes data as they are executed on the trading system. Reports of new equity trades, of which there were approximately 55,000, have been extracted and converted to the FIX market data incremental refresh structure, similar to that shown in Figure 3. The number of MDEntry items in each message

is varied from 1–10 to alter the message size and allow estimation of the fixed and incremental performance costs for each message.

4.2 Software

Application data structures were translated to and from the wire formats using schema–specific encoders and decoders. The software tools used to accomplish this were as follows:

- SOAP messages were produced using gSOAP [29], a free, high performance toolkit. Studies [29, 5] showed that gSOAP had substantially higher performance than some commonly used SOAP implementations, although not as high as another special–purpose research implementation. The XML schema for the message was based on the FIXML DTD [10], with encoders and decoders generated in C++.

- An implementation of a subset of the FIX protocol was developed specifically for the research. The application structure was defined in CORBA IDL, and the encoders and decoders were in C++.

- The same CORBA IDL definition as used with the FIX implementation was compiled using the TAO CORBA ORB's [24] IDL compiler to generate C++ encoders and decoders for the CDR wire format.

4.3 Environment

The latency and throughput tests were run over both 10 Mbps and 100 Mbps Ethernet. Communication between real–time trading systems often occurs across leased lines or the public Internet, and so 10 Mbps may be more comparable to the available bandwidth in these cases.

The client system was a uniprocessor 900 MHz Pentium 3 with 256 MB of RAM, 256 KB level 2 cache and running Windows 2000. The test software for this system was compiled using the Borland C++ 5.6.1 compiler.

The server was a uniprocessor 500 MHz Pentium 3 with 256 MB of RAM, 512 KB level 2 cache, running Redhat Linux 7.3 with the 2.4.18-3 kernel. The test software for this system was compiled using the g++ 3.2.1 compiler.

5 Experimental Results

5.1 Message Size

Prior to running experiments to measure latency and throughput, data were collected to compare the size of the application messages when encoded in each of the SOAP, FIX and CDR wire formats.

Table 1: Summary of message size costs

	Fixed Cost	**Per–Entry Cost**
FIX	81.93 bytes	109.9 bytes
CDR	84.90 bytes	208.1 bytes
SOAP	639.0 bytes	386.9 bytes

Figure 4 and Table 1 show the message sizes of the application data structure when encoded into each format. The figure shows that SOAP messages are substantially larger, being some 4–5.5 times larger than the equivalent FIX message, and 2–3.5 times larger than one in encoded using CDR. This is on the low side of the relative size results presented by existing SOAP and XML performance studies [4, 5, 14].

Figure 5 shows the application structure from Figure 3 encoded using FIX, and Figure 6 shows the same data encoded into SOAP. Here we see that the XML namespaces, the more verbose tag

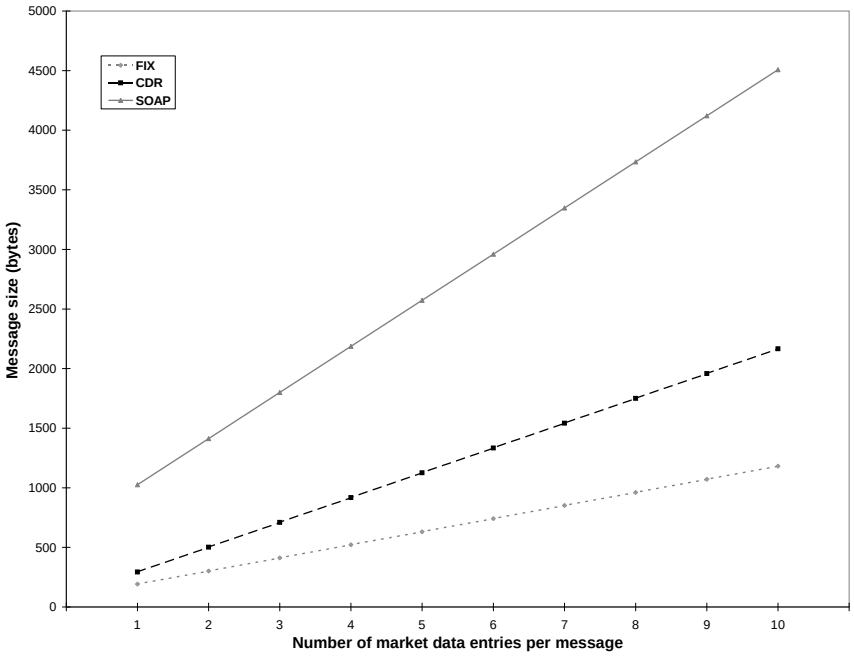

Figure 4: Message size comparison of the wire formats

```
8=FIX.4.3●9=00000163●35=X●49=ABC●56=XYZ●34=1234●
52=20021128-18:09:42●268=1●279=0●269=2●278=T07822
4●55=CBG●461=ESXXXX●270=6.35●271=451●272=20021
125●273=10:18:58●287=3●288=2402●289=3713●10=009●
```

Figure 5: FIX representation

```
<?xml version=”1.0” encoding=”UTF-8”?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
xmlns:SOAP-ENC=”http://schemas.xmlsoap.org/soap/encoding/”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
<SOAP-ENV:Body
SOAP-ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”>
<sendMessage>
<FIXMLMessage>
<Header>
<Sender><CompID>ABC</CompID></Sender>
<Target><CompID>XYZ</CompID></Target>
<SendingTime>2002-11-28T18:09:42.000</SendingTime>
</Header>
<ApplicationMessage>
<MarketDataInc>
<MDIncList>
<MDIncGroup>
<MDUpdateAction>0</MDUpdateAction>
<MDEntryID>T078224</MDEntryID>
<Instrument>
<Symbol>CBG</Symbol>
<CFICode>ESXXXX</CFICode>
</Instrument>
<MDEntryPx>6.35</MDEntryPx>
<MDEntrySize>451</MDEntrySize>
<MDEntryDate>2002-11-25</MDEntryDate>
<MDEntryTime>10:18:58.000</MDEntryTime>
<SellerDays>3</SellerDays>
<MDEntryBuyer>2402</MDEntryBuyer>
<MDEntrySeller>3713</MDEntrySeller>
</MDIncGroup>
</MDIncList>
</MarketDataInc>
</ApplicationMessage>
</FIXMLMessage>
</sendMessage>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6: SOAP representation

names and syntax contribute to the SOAP message being substantially larger.

```
union Optional_String switch(boolean) {
case TRUE:
    string value;
};
```

Figure 7: Optional field idiom used for application data structure definition in CORBA IDL

Figure 4 also shows that FIX has a more compact wire representation than CDR, which runs counter to what is expected. CDR is larger due to CORBA IDL's lack of built–in support for optional fields. We have instead used a common idiom for defining optional fields in CORBA [16], as illustrated by Figure 7. This means that each optional field uses a single–byte indicator to show whether it is present or not. As the message header contains approximately 20 optional fields, and as each market data entry contains about 70, then with most of these fields unset there is considerable overhead. Alternative binary wire formats with true support for optional fields, such as ASN.1/BER, may offer more compact messages.

5.2 Latency

Table 2: Summary of round–trip costs over 10 Mbps network

	Fixed Cost	**Per–Entry Cost**
FIX	1.032 msec	0.2516 msec
CDR	1.094 msec	0.4017 msec
SOAP	2.059 msec	1.176 msec

Figure 8 and Table 2 present the measurements for round–trip times over a 10 Mbps network. This shows that FIX and CDR have similar fixed costs, but that CDR has about 60% higher cost per entry. Although the fixed cost for SOAP is only double that of the others, its per–entry cost is nearly 5 times that of FIX, and 3 times that for CDR.

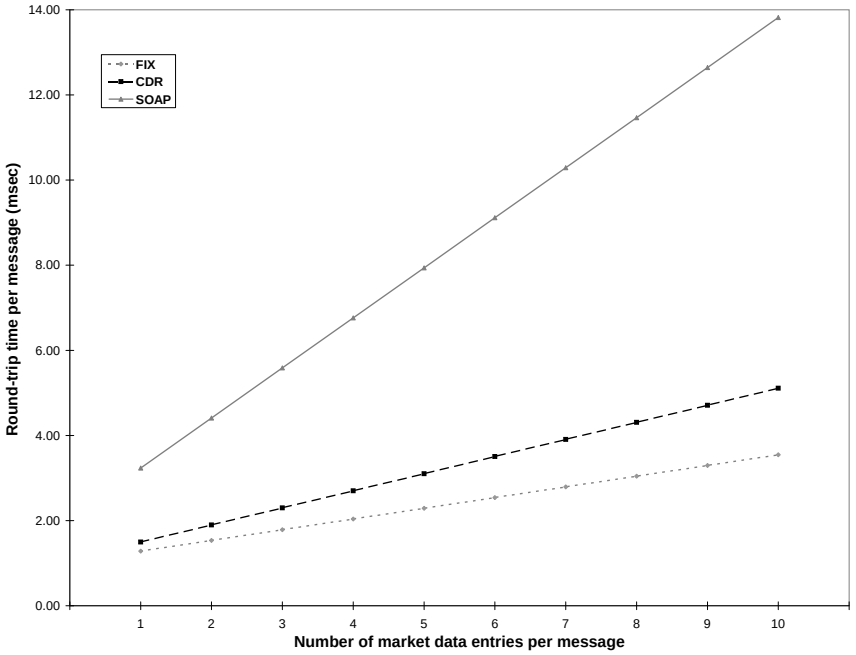

Figure 8: Round–trip times over 10 Mbps network

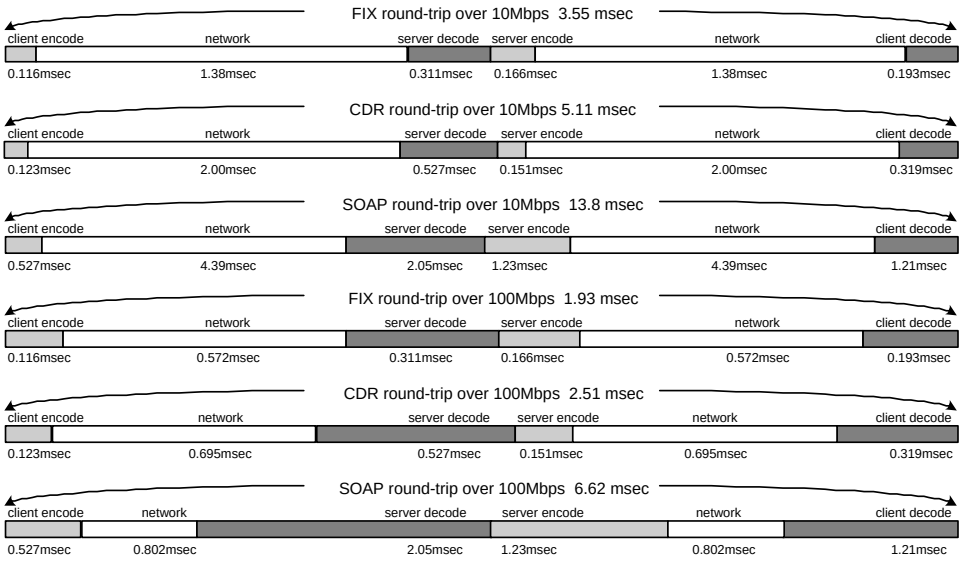

Figure 9: Cost breakdown for round–trip times for messages with 10 market data entries

The breakdown of costs in Figure 9 shows that for all three wire formats, over a 10 Mbps network, the largest cost is the time spent on the network. This would suggest that in this environment the size of the message on the wire is the major limiting factor. Over the slower network, FIX's more compact message representation contributes to its lower round–trip times than CDR.

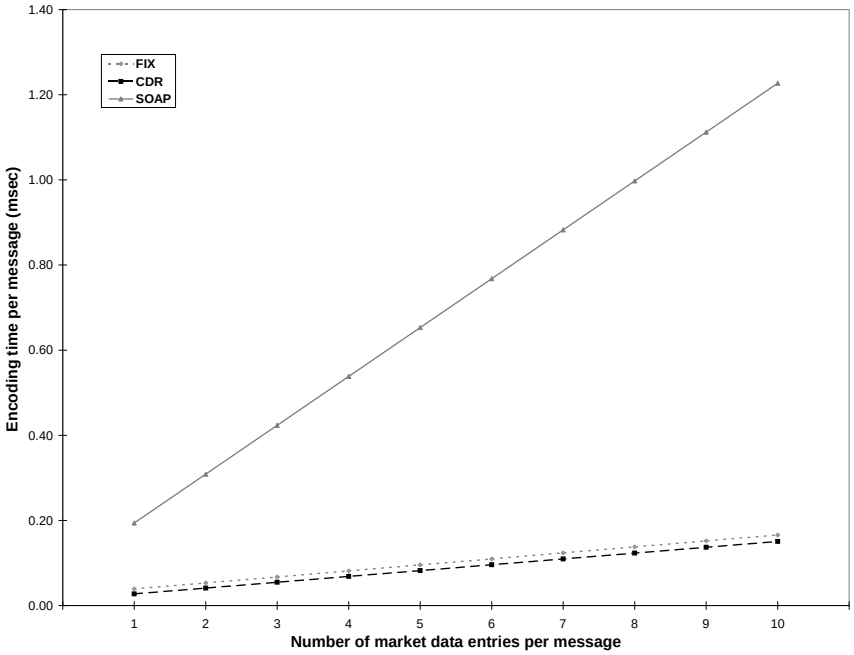

Figure 10: Server–side encoding costs

Table 3: Summary of server–side encoding costs

	Fixed Cost	**Per–Entry Cost**
FIX	0.0250 msec	0.0141 msec
CDR	0.0136 msec	0.0137 msec
SOAP	0.0791 msec	0.1148 msec

Over a 100 Mbps network, time spent on the network is less significant in overall round–trip times. Figure 10 and Table 3 show the encoding costs for the wire formats, and Figure 11 and Table 4 show the relative decoding costs. For 100 Mbps Ethernet, the substantially higher

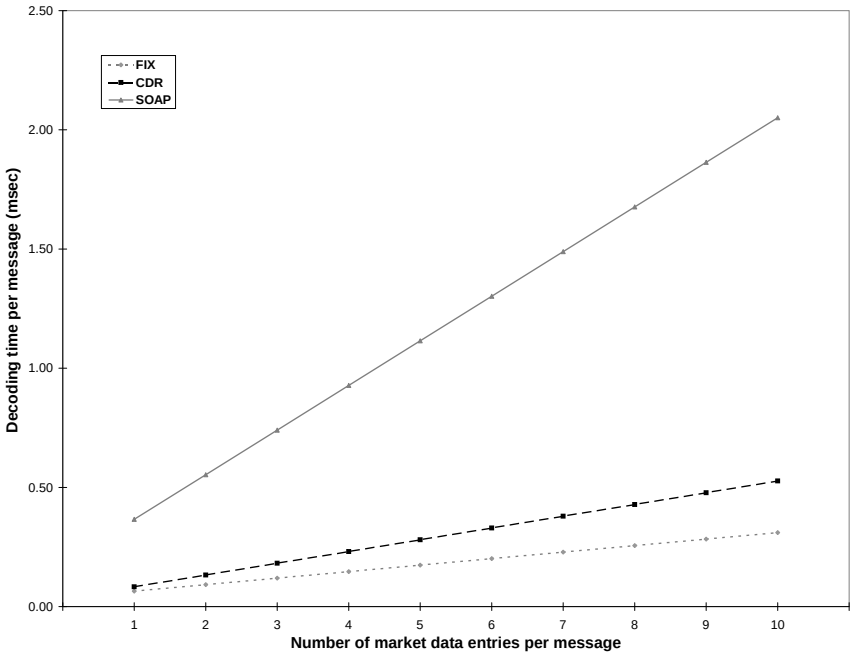

Figure 11: Server–side decoding costs

Table 4: Summary of server–side decoding costs

	Fixed Cost	**Per–Entry Cost**
FIX	0.0375 msec	0.0273 msec
CDR	0.0339 msec	0.0493 msec
SOAP	0.1789 msec	0.1872 msec

encoding and decoding costs for SOAP contribute most to its poorer performance, with round–trips considerably more than twice as expensive as FIX or CDR.

An interesting result shown in Figure 11 is that FIX, a text–based wire format, has lower decoding costs than CDR, a binary format. This is particularly significant given the greater complexity involved in decoding FIX, with the presence of the tags in the wire format, flexible field ordering, and the fact that many fields may or may not be present at all on the wire. With CDR, on the other hand, all fields would be decoded in a fixed order as determined by their definition in the CORBA IDL. This result suggests two things:

- With this realistic application message content there is a mix of string, integer and floating point values. The cost of converting numerical data from text to binary, identified as major by other studies, does not have a predominant role.
- The cost of handling the complexity of the FIX message structure is minor compared to the cost of decoding a field, and that greater benefit is derived from not having to process the large number of optional fields that are not present on the wire. With the message encoding, encoders for all wire formats must test the presence of every field, and the advantage is lost.

5.3 Throughput

Figure 12 displays the measurements for throughput over a 10 Mbps network. For this slower network configuration, the network itself was observed to be the bottleneck for all three wire formats. As with latency, this result suggests that in an environment with lower bandwidth, the size of the message is the major factor affecting performance. This allows FIX, with the most compact messages, to achieve the highest throughput values.

For 100 Mbps networks, the CPU on the slower server machine was observed to be the bottleneck, and consequently the network was under utilised. The results in Figure 13 show that FIX

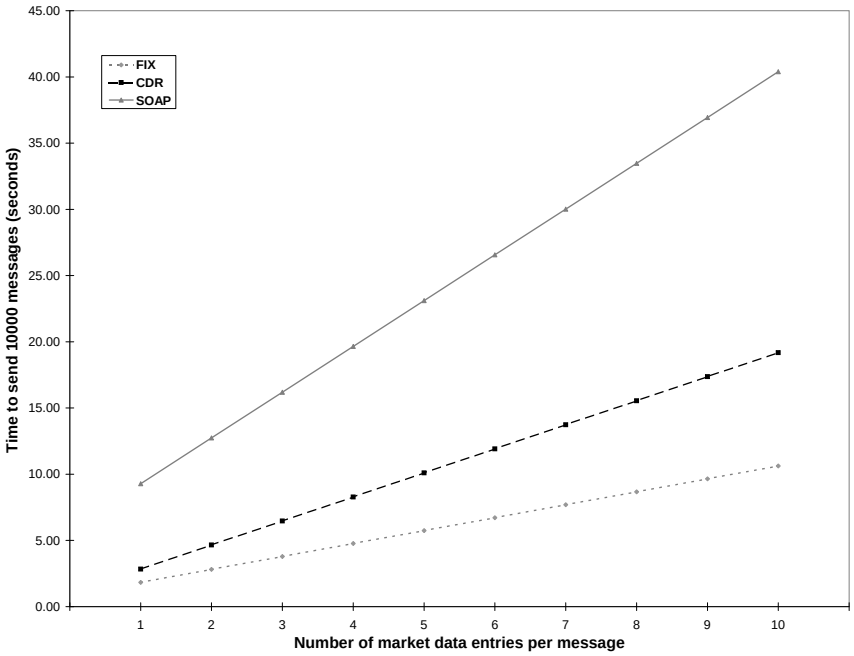

Figure 12: Throughput over 10 Mbps network

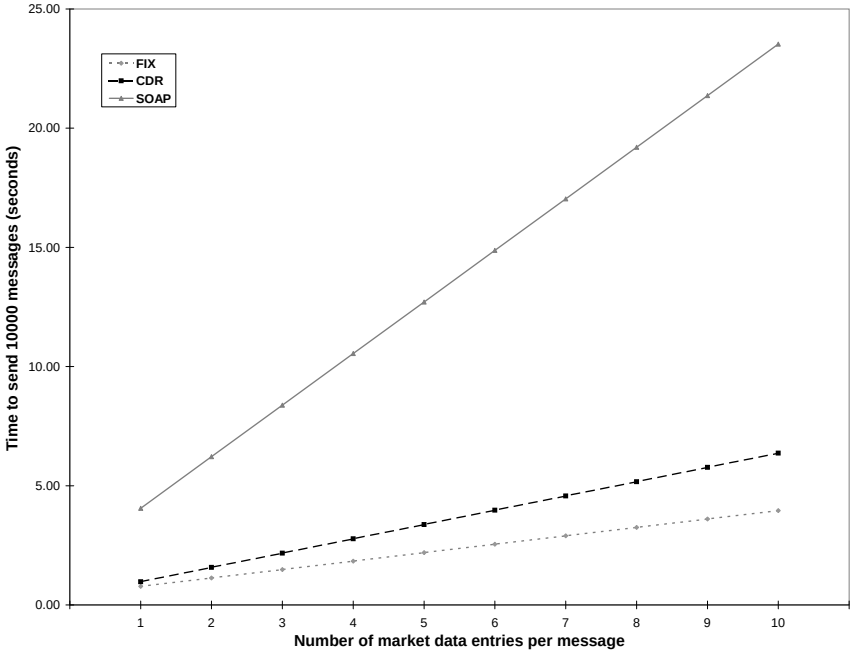

Figure 13: Throughput over 100 Mbps network

again achieved the highest throughput, although CDR has lower encoding costs. This is a result of the decoding, which had a lower cost for FIX than CDR, being performed on the slower machine. Reversing the roles of the machines changes the relative throughput of the wire formats.

Interestingly, the throughput performance of SOAP relative to the other two wire formats is worse for the 100 Mbps network. This is due to the ratio of SOAP decoding cost to FIX or CDR being greater than the equivalent ratio for message size.

5.4 Compact XML Tags

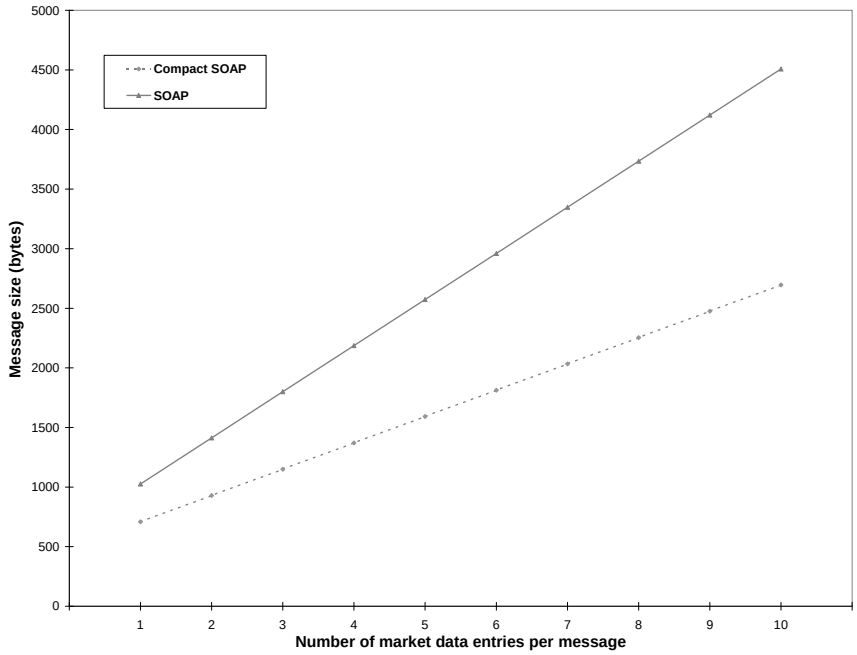

Figure 14: Message size reduction when compact XML tags are used

An alternative method for reducing the size of the SOAP messages investigated in this study was to reduce the length of the XML tag names. This was done by replacing the FIXML names with short 2–4 character strings based on the numeric FIX tags. This reduced the size of the

SOAP messages by approximately 30–40% as shown in Figure 14, but clearly sacrifices message readability in favour of the potential performance gains.

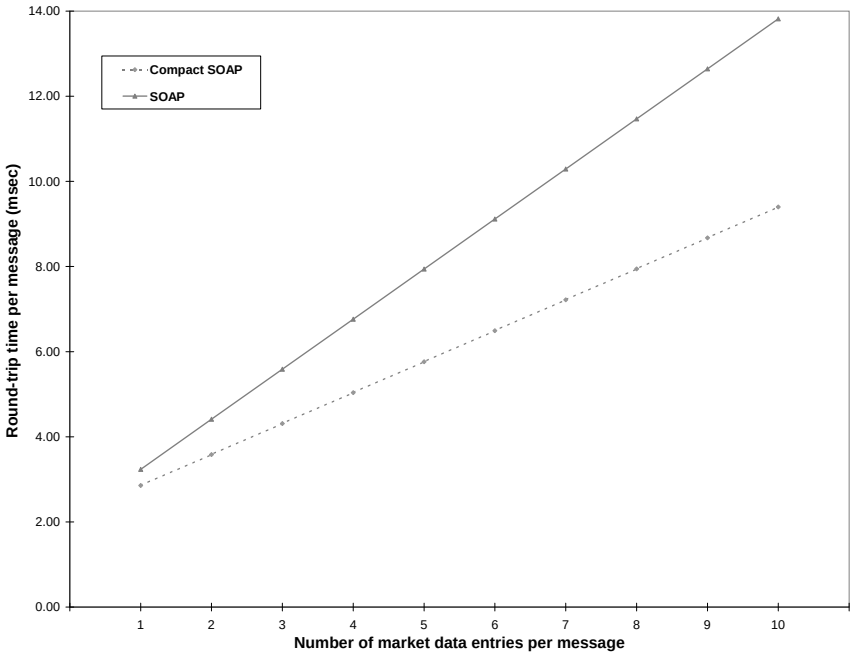

Figure 15: Round–trip times for compact SOAP messages over 10 Mbps network

Figure 15 shows that the more compact SOAP messages do provide gains in performance over 10 Mbps Ethernet, where the time on the network is the major cost. However, the performance improvement is not in the same proportion to the reduction in message size. When considering the relative decoding costs shown in Figure 16, we see that there is not a commensurate improvement in decoding performance. Furthermore, the use of compact SOAP has a negligible effect on encoding efficiency. This suggests that the major cost of the XML encoding and decoding is in the structural complexity and syntactic elements, rather than the data contained in the message or the tag names.

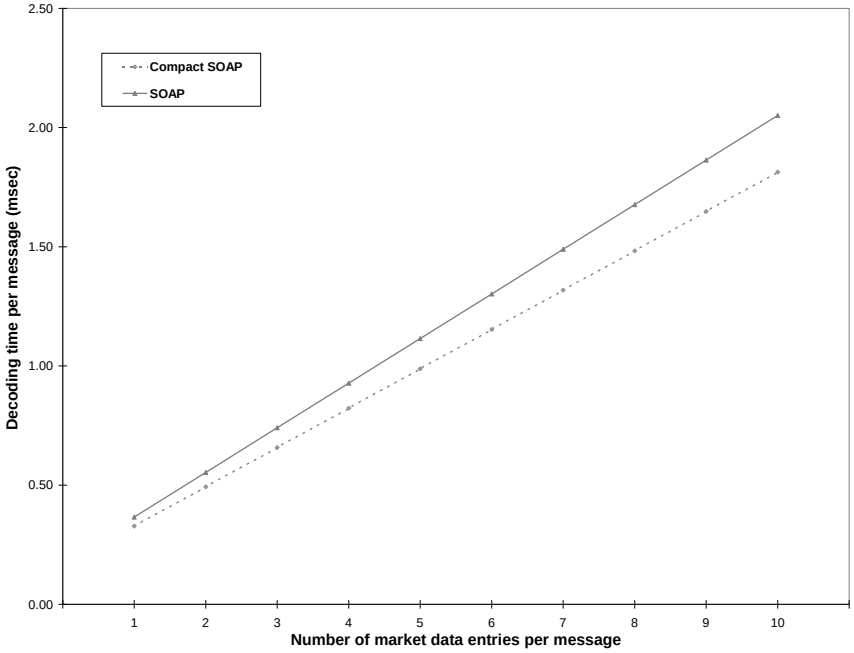

Figure 16: Decoding costs for compact SOAP messages

6 Performance Requirements and SOAP

Only a single day of actual trading data from the Australian Stock Exchange is used in the research, and consequently the results cannot be extrapolated for the system's operation in general. However, given the low level of variation in the results, in practice it is likely to be a reasonable indication of performance for similar message data from other days.

Given that, this section attempts to assess whether the SOAP implementation, as benchmarked, can meet the performance requirements of the particular day's data. The instantaneous peak number of trade report messages per second in this data is 70. For simplicity, the calculations will only consider whether the SOAP implementation can maintain a sustained load of 70 trades per second.

The throughput results presented in this paper show that SOAP can clearly meet the requirement of 70 trades per second. Even the least efficient scenario, with one trade report per message,

exceeds the required performance by 15 times.

However, this calculation will instead use the results to determine:

- What is the minimum bandwidth required for the wide area network connecting the client to the exchange to be able to sustain 70 trade reports per second.
- How many concurrent client connections could be supported by an exchange computer system of similar specification to the server used in this research.

Consider a hypothetical system for transmitting trade reports, similar to that shown in Figure 1, where:

- The messages are encoded using SOAP.
- The messages are sent in only one direction, from exchange to client, similar to the throughput benchmark tests.
- To minimise bandwidth utilisation, the exchange buffers trade report entries for up to 1 second, or until it has 10 entries. When either condition is reached it immediately sends a message with at most 10 entries.

With an instantaneous peak of 70 messages per second, the buffering strategy of the hypothetical system would result in the transmission every second of seven messages containing 10 trade reports each. Using the results from Table 1, a single 10–entry trade report message encoded in SOAP requires 4508 bytes. Therefore, every second would involve the transmission of 31556 bytes of data. This is approximately equivalent to needing a network with 256 Kbps bandwidth.

With respect to the hardware requirements, the encoding costs are 1.23 msec per 10–entry message. To sustain seven 10–entry messages per second, a server at the exchange that is encoding

and transmitting the trade reports would be utilising less than 1% of the CPU, and could transmit to roughly 100 brokers simultaneously.

7 Conclusions

Earlier studies into SOAP and XML performance [4, 5] found that the conversion from text to binary and vice versa was the major cost, especially the costs associated with encoding and decoding floating point values. However, these studies were oriented towards the application of SOAP and XML to scientific computing, with test data consisting for the large part of numerical values. This research, however, attempts to study the performance of SOAP using realistic business application messages, with capital markets trading systems as the context.

By benchmarking the performance of SOAP against FIX and CDR, the study finds that:

- *SOAP performs poorly compared to the binary wire format, CDR.* SOAP messages are some 2–3.5 times the size of the equivalent messages encoded using CDR, latency over local networks is more than doubled, and encoding and decoding messages is 4–8 times more expensive.

- *SOAP performs poorly when compared to FIX.* SOAP messages are observed to be some 4–5.5 times the size of the equivalent FIX message. The latency costs on local networks are increased by a factor of 2–3 times, and encoding/decoding costs are up to eight times higher.

- *FIX performs comparably to CDR.* Given that the overall performance of FIX, with its text–based wire format, was comparable to CDR — and in fact outperforming it for decoding — it is clear that conversion of text–to–binary and back need not be a major factor affecting performance in business scenarios. This conclusion differs from the earlier studies that focused on scientific applications.

When designing performance–conscious business systems, it is important to consider the environment in which a system will be deployed. Although for fast networks the speed of encoding and decoding is the predominant determining factor, for slower networks it is the size of the encoded message that determines both latency and throughput performance. The latter is important for business–to–business integration which, in capital markets as in most other domains, often occurs over wide area networks.

An important conclusion drawn from the benchmark results is that SOAP's poor performance is not adequately explained simply by the disadvantages of text–based over binary wire formats. One potential cause investigated in this study is the greater syntactic and structural complexity in XML messages. Simply reducing the size of the encoded SOAP messages by using shorter tags did not improve the speed of encoding and decoding in proportion to the reduction in message size. This indicates that the greater cost associated with encoding and decoding the SOAP messages may be due to the complexity of XML syntax and its rich on–the–wire structure.

Further research and experimentation with alternative schema definitions may be useful in clarifying the impact of message structure on performance. An outcome of such research could include general advice on how to design schema definitions for SOAP with performance requirements in mind.

Finally, the study finds that SOAP may be used as it is in certain small–to–medium real–time trading applications. Using the real trading data supplied by the Australian Stock Exchange, the research findings suggest that SOAP can adequately cover the peak throughput requirements for that system. However, the potential problem with SOAP in these cases is that scaling to larger numbers of clients will have proportionally greater hardware and network bandwidth needs than either FIX or CDR.

8 Acknowledgments

The authors would like to gratefully acknowledge the assistance of the Australian Stock Exchange in supplying one day of “Signal C” trading data.

References

[1] Australian Stock Exchange. The SEATS computer system, 2000. `http://www.asx.com.au/markets/l4/SEATS_AM4.shtm`, accessed 1 June 2002.

[2] Australian Stock Exchange. Market Information Signal C Manual: Public Progressive Intra-Day Signal, September 2002.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol - HTTP/1.0, 1996. IETF RFC 1945, `http://www.ietf.org/rfc/rfc1945.txt`.

[4] F. E. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener. Efficient wire formats for high performance computing. In *Proceedings of the 2000 Conference on Supercomputing*, 2000.

[5] K. Chiu, M. Govindaraju, and R. Bramley. Investigating the limits of SOAP performance for scientific computing. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, pages 246–254, 2002.

[6] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: An introduction to SOAP, WSDL, UDDI. *IEEE Internet Computing*, 6(2):86–93, March-April 2002.

[7] D. Davis and M. Parashar. Latency performance of SOAP implementations. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 407–412, 2002.

[8] M. Fan, J. Stallaert, and A. B. Whinston. The internet and the future of financial markets. *Communications of the ACM*, 43(11):83–88, November 2000.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol - HTTP/1.1, 1999. IETF RFC 2616, `http://www.ietf.org/rfc/rfc2616.txt`.

[10] FIX Protocol Ltd. FIXML: A markup language for the FIX application message layer. http://www.fixprotocol.org/WORKGROUPS/928951581/wpaper.html, accessed 8 June 2002.

[11] FIX Protocol Ltd. The Financial Information Exchange Protocol (FIX), version 4.3, August 2001. http://www.fixprotocol.org/specification/fix-43-pdf.zip, accessed 8 June 2002.

[12] M. Girardot and N. Sundaresan. Millau: An encoding format for efficient representation and exchange of XML over the web. In *Proceedings of the 9th International World Wide Web Conference*, pages 747–765, 2000.

[13] J. Goeller. FIXML and STP related efforts, 2000. http://www.fixprotocol.org/WORKGROUPS/928951581/XML_STP_John6.ppt, powerpoint presentation, accessed 8 June 2002.

[14] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, and D. Gannon. Requirements for and evaluation of RMI protocols for scientific computing. In *Proceedings of the 2000 Conference on Supercomputing*, 2000.

[15] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Sams Publishing, Indianapolis, 2002.

[16] M. Henning and S. Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, Reading, Massachusetts, 1999.

[17] V. Jacobson. Congestion avoidance and control. In *Symposium proceedings on Communications architectures and protocols*, pages 314–329. ACM Press, 1988.

[18] C. Kohlhoff and R. Steele. Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems. In *Alternate proceedings of the Twelfth International World Wide Web Conference*, pages 262–270, May 2003.

[19] B. Liu and E. A. Fox. Web traffic latency: Characteristics and implications. *J.UCS: Journal of Universal Computer Science*, 4(9):763–778, 1998.

[20] B. Martin and B. Jano. WAP binary XML content format, June 1999. http://www.w3.org/TR/wbxml/, accessed 1 June 2002.

[21] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley. Network performance effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 155–166, 1997.

[22] Object Management Group. The Common Object Request Broker Architecture: Core Specification, version 3.0, November 2002.

[23] F. A. Rabhi and B. Benatallah. An integrated service architecture for managing capital market systems. *IEEE Network*, 16(1):15–19, 2002.

[24] D. C. Schmidt, D. L. Levine, and S. Mungee. The design of the TAO real-time object request broker. *Computer Communications*, 21(4):294–324, April 1998.

[25] S. E. Spero. Analysis of HTTP performance problems, 1994. `http://www.w3.org/Protocols/HTTP/1.0/HTTPPerformance.html`, accessed 15 June 2002.

[26] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Reading, Massachusetts, 1994.

[27] N. Sundaresan and R. Moussa. Algorithms and programming models for efficient representation of XML for internet applications. In *Proceedings of the 10th International World Wide Web Conference*, pages 366–375, 2001.

[28] SWIFT. About SWIFT - History. `http://www.swift.com`, accessed 3 June 2002.

[29] R. A. van Engelen and K. A. Gallivan. The gSOAP toolkit for web services and peer-to-peer computing networks. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 128–135, 2002.

[30] P. Widener, G. Eisenhauer, and K. Schwan. Open metadata formats: Efficient XML-based communication for high performance computing. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, pages 371–380, 2001.