

An Evaluation of Contemporary Commercial SOAP Implementations

Alex Ng¹

alexng@ics.mq.edu.au

Shiping Chen²

shiping.chen@csiro.au

Paul Greenfield²

paul.greenfield@csiro.au

¹Department of Computing, Macquarie University, North Ryde, NSW 2109, Australia

²CSIRO ICT Centre, PO Box 17, North Ryde, NSW 1670, Australia

Abstract

SOAP is now widely accepted as the core transport protocol for Web Services but there are still open questions about whether SOAP can really meet the performance needs of business. This paper attempts to address some of these questions by evaluating and comparing the performance of several contemporary commercial SOAP implementations. The reported tests send a variety of SOAP messages to Web Services implemented using a number of current commercial SOAP implementations and measure several aspects of the Web Service's performance. This paper presents a view of the current state of SOAP performance and gives some insight into the limitations of today's SOAP implementations.

1. Introduction

The computing industry has adopted Web Services as its common middleware technology for the purposes of integrating enterprise applications over the Internet and supporting loosely-coupled Service Oriented Architectures (SOA). As defined in the W3C Web Services Architecture Requirements [1], client systems 'interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols'. The combination of loosely-coupled architectures and text-based messaging gives Web Services the capability to allow heterogeneous systems and applications that are programmed in different languages and technologies to interoperate with one another [2].

Figure 1, taken from [7], shows the Web Services architecture stack and the relationships between different specifications/services. SOAP provides the fundamental messaging infrastructure used by the other components and services of Web Services, supporting XML document exchange and Remote Procedure Calls (RPC) using XML messages.

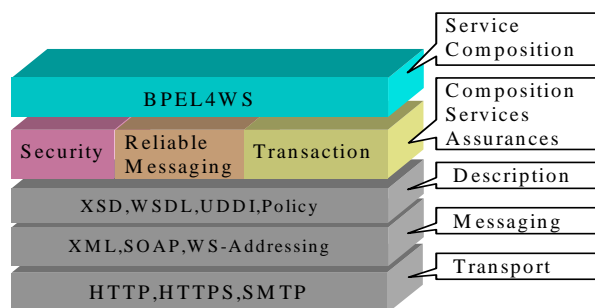


Figure 1. Web Services protocol architecture

SOAP is based on XML technology, and so inherits the inefficiencies of XML, requiring more bandwidth, more storage and more processing power than equivalent binary implementations. The performance of SOAP has been reported in earlier studies [3-6, 8, 9], but we consider that these papers do not adequately address the basic questions of concern to the commercial world:

- (1) What level of performance is shown by major commercial Web Services implementations?
- (2) How does the performance of SOAP compare to that offered by conventional non-SOAP technologies?

In this paper, we provide some answers to these questions by implementing the same set of Web Services with a number of commercial SOAP and non-SOAP implementations and measuring the performance against the same series of tests.

The rest of this paper is organised as follows: a brief discussion of the SOAP protocol is given in section 2. Section 3 explains how we conducted the evaluation exercise, including the decisions we made and our metrics. Section 4 provides the results of our analysis of the data collected in our measurements. Section 5 discusses related work and our conclusions are presented in section 6.

2. Background

SOAP is a lightweight, stateless, communication protocol which lets applications exchange structured textual messages across networks, especially across the Internet. A SOAP message has a simple structure: an envelope containing a header and a body. The body carries the message payload, such as order and payment details, and the header is an optional component used to hold control information, such as required security policies. SOAP is designed to be independent from network protocols, but most current SOAP implementations use the HTTP binding due to its wide availability and ability to go through firewalls. Figure 2 shows a SOAP message bound to HTTP.

```
POST /ROLMAN/ROLMANws.asmx HTTP/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://hostname/
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope>
  <soap:Header>
  <\soap:Head>
  <SOAP:Body>
  <app:CustomerServices xmlns:app=...>
    <Customer>
      <Name>.....<\Name>
      <Address>.....<\Address>
      <Phone>.....<\Phone>
    <\Customer>
  <\app:CustomerServices>
<\soap:Body>
<\soap:Envelope>
```

Figure 2. HTTP SOAP message binding

There are two styles of writing SOAP messages:

- (a) *document* and
- (b) *RPC* – Remote Procedure Call.

There are also two different techniques for serialising the data in the SOAP body:

- (a) using *literal* XML Schema definitions and
- (b) using the *SOAP encoding* rules.

This gives four possible combinations or ‘encoding styles’: *document/literal*, *document/encoded*, *RPC/literal*, and *RPC/encoded*. The *RPC style* was widely adopted initially due to its familiarity and the easy mapping of traditional distributed computing technologies onto SOAP messaging infrastructure. *Document style* SOAP messaging reflects a more native use of XML and is more flexible than the *RPC style*. The *literal* encoding technique uses standard XML schemas to define how content is encoded, while the *encoded* approach uses the SOAP-specific

encoding rules that were defined in earlier SOAP standards.

The Web Services Interop (WS-I) Basic Profile [2] prefers (in R1005 to R1007) ‘the use of literal, non-encoded XML’. Furthermore, SOAP version 1.2 [10] makes the support of the SOAP encoding rules and RPC conventions optional. Although, *document/literal* has become the preferred SOAP messaging and encoding styles, we included other styles in this study, allowing us to determine if the widespread adoption of *document/literal* is a good choice on performance grounds.

3. Experimental Platform and Design

One of our goals in this study was to determine the general level of performance available from commercial SOAP implementations. We designed a set of scenarios that reflected typical commercial usage and used these to test a number of SOAP implementations using different messaging and encoding styles. We chose three commercial SOAP Web Services implementations from three leading vendors. These commercial products are only identified in our evaluation as products A, B and C due to the benchmarking restrictions imposed by some licence agreements.

Assuming that these are good implementations of the SOAP standards, these tests will give us insight into the inherent performance of SOAP running over HTTP. This performance was then compared to an equivalent non-SOAP protocol running directly over TCP/IP and using a binary formatter (Microsoft .NET Remoting). This comparison gives us a standard for comparing performance, allowing us to see how well SOAP performs in an absolute sense, rather than just comparing SOAP implementations against other SOAP implementations.

3.1 Overall Experimental Design

The test scenarios were designed to resemble real life commercial transactions while avoiding any unnecessary server-side processing overhead which may affect the measured performance, such as database accesses and business calculations. Each test setup consisted of the following:

1. A target Web Service provider that receives the test message, decodes the message to access the parameter values and returns the original

message with its header value changed to “RESPONSE”.

2. A multi-threaded Web Service test driver written in Microsoft’s C# language. The same test driver was used throughout all of our experiments to ensure that the measured performance figures only reflect differences in the inherent performance of the server implementation under test.

A configuration file was used to control the setup of the test environment, specifying parameters such as the number of client threads to be used, the duration of the test run, and the messaging and SOAP encoding styles to be used.

3.1.1 Test Message

We used three test messages in this study, allowing us to see how well the implementations under test handled messages of varying length and complexity. The first message type (*simple*) contains a single customer’s account record and used string, Boolean and datetime data items. The second message type (*medium*) consists of twenty customer account records, perhaps representing a typical batch inquiry and subsequent update transaction. The third type of message (*complex*) consists of one customer record and 50 product details, perhaps representing an invoice or a customer statement.

<pre>public class customer { string customerId; string password; string firstName; string lastName; string address; string phoneNum; string email; string accountNo; bool isValid; DateTime lastModified; }</pre>	<pre>public class product { long categoryId; bool isAvailable; bool doWeBuy; bool doWeSell; long productId; string description; double price; long quantity; string quantityUnit; long pricePolicyId; DateTime ETA; DateTime lastModified; }</pre>
---	--

Figure 3. Customer and Product structures

3.2 Performance Metrics and Measurement

The following performance metrics were used to evaluate the performance of the SOAP implementations under test:

- **Latency.** We define latency as the round-trip time taken to send and receive a single message, from the test driver to the server and back to the waiting test driver. High-resolution timers were used to measure the time taken for each round-trip. The test was repeated for the period of time specified in the configuration file.
- **Throughput.** A set of tests was conducted to find the peak throughput available from each SOAP implementation. These tests use a number of concurrent driver threads to send large volumes of concurrent messages to the SOAP servers being tested and record the number of round-trips completed each second. The performance results reflect the level of resources needed to process each SOAP message and the impacts of any internal lock contention inside the SOAP servers. Each client thread went through an initialisation period and then synchronised with all other client threads before starting to send test messages to the server, so ensuring that the concurrent driver threads were all actually sending requests at the same time.
- **Serialisation and deserialisation overheads.** Chiu, Govindaraju, and Bramley report in [4] that XML serialisation and deserialisation are the bottlenecks in SOAP processing. In order to quantify the SOAP serialisation and deserialisation overheads, we conducted a separate set of tests using the *SOAP extension* mechanism provided by Microsoft’s ASP.NET product. These tests captured high resolution timestamps at different stages of the message handling process as shown in Figure 4. These timestamps were then used to calculate the cost of XML serialisation using the following equations:
 - $T2 = \text{time spent in deserialisation, } (AfterDeserialise - BeforeDeserialise)$
 - $T3 = \text{time spent in application logic, } (BeforeSerialise - AfterDeserialise)$
 - $T4 = \text{time spent in serialisation } (AfterSerialise - BeforeSerialise)$
 - $T1 + T5 = \text{Latency} - (T2+T4)*2 - T3 = \text{network time}$

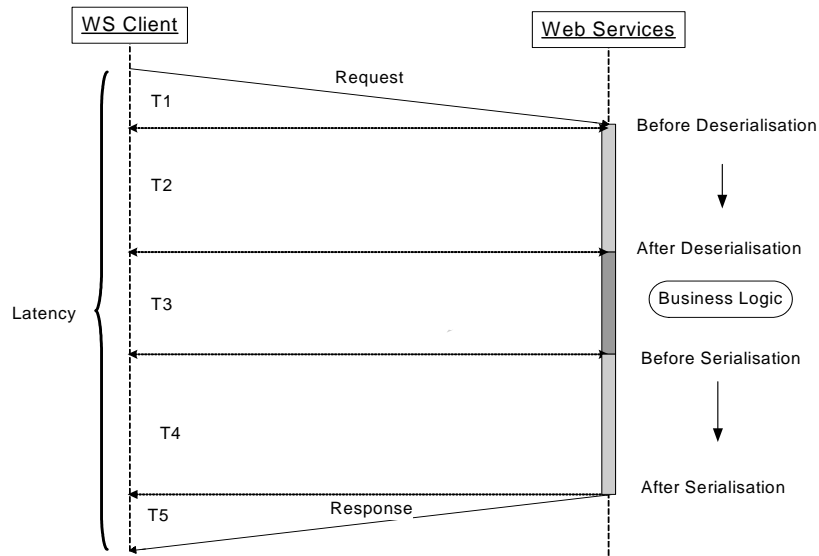


Figure 4. Serialisation and deserialisation measurements

4. Performance Analysis

Identical Dell 6650 servers were used as both client and server systems. The hardware and software configuration of these systems was:

- Four 1.6GHz Intel Xeon MP processors
- 3669Mbytes of memory
- Microsoft Windows 2000 Advanced Server with Service Pack 3, build 2195
- Microsoft .NET Framework 1.1

All the experiments were run over a 100Mbps switched Ethernet LAN using a 100Mbps Cisco switch. Microsoft Visual Studio.NET and C# were used to develop the test driver that was used for all tests.

All of the implementations under test supported only a limited subset of the possible combinations of SOAP encoding and messaging styles. These restrictions are summarised in Table 1.

4.1 Message Size Analysis

A shareware TCP traffic monitor program (tcptrace) [11] was used to measure the size of the SOAP messages passed from client to server. Table 2 shows the results for the different message types for the implementations under test.

Product	Doc/Lit	Doc/Enc	RPC/Lit	RPC/Enc
Product A	✓	✓	✗	✓
Product B	✓	✗	✓	✓
Product C	✗	✗	✗	✓

Table 1. Encoding styles supported by each SOAP implementation

Product	Encoding Style	Simple Msg (byte)	Med. Msg (byte)	Cmplx Msg (byte)
Product A	Doc/Lit	873	7581	21392
	Doc/Enc	1593	13973	38842
	RPC/Enc	1524	13969	38838
Product B	Doc/Lit	870	7513	21566
	RPC/Enc	1597	14035	38999
Product C	RPC/Enc	1664	14566	40273

Table 2. SOAP message sizes for different implementations

Document/Literal encoding style is clearly the most efficient for all implementations, requiring only half the number of bytes for all messages compared to all uses of the SOAP encoding rules. All products produced very similar-sized messages for any given encoding style.

4.2 Latency Analysis

The latency tests were conducted using a single thread in the test driver to repeatedly send requests to the server under test and to wait for the response, and measuring the response time using high resolution timers. Figure 5 shows the results obtained for the entire set of latency tests, for all products and all variants of message type, size and encoding styles.

4.2.1 Simple Message Latency

The latency results for simple messages show that the Document/Literal combination generally offers better latency time (4mS) than the other combinations of message style and encoding (RPC/Encoded ranges from 5mS to 16mS, and Document/Encoded gives 5mS). The worst performer was Product C RPC/Encoded (16mS). The non-SOAP .NET Remoting test showed a response time of 1mS, indicating that the fastest SOAP implementation is still significantly slower than using binary encoding and direct TCP/IP transport, although at 4-5mS SOAP may

still meet the performance requirements of many applications.

4.2.2 Medium Message Latency

The two Document/Literal implementations are also the performance leaders for medium messages, although a performance gap is starting to open up between Products A and B (10mS to 17mS). The gap between Document/Literal and the other encoding styles has also widened, with the RPC/Encoded combinations showing the poorest performance at between two and four times slower than the fastest Document/Literal implementation. .NET Remoting returns a latency result of 7mS, meaning that the relative gap between this binary/TCP protocol and the best SOAP/HTTP implementation has narrowed (7mS vs. 10mS).

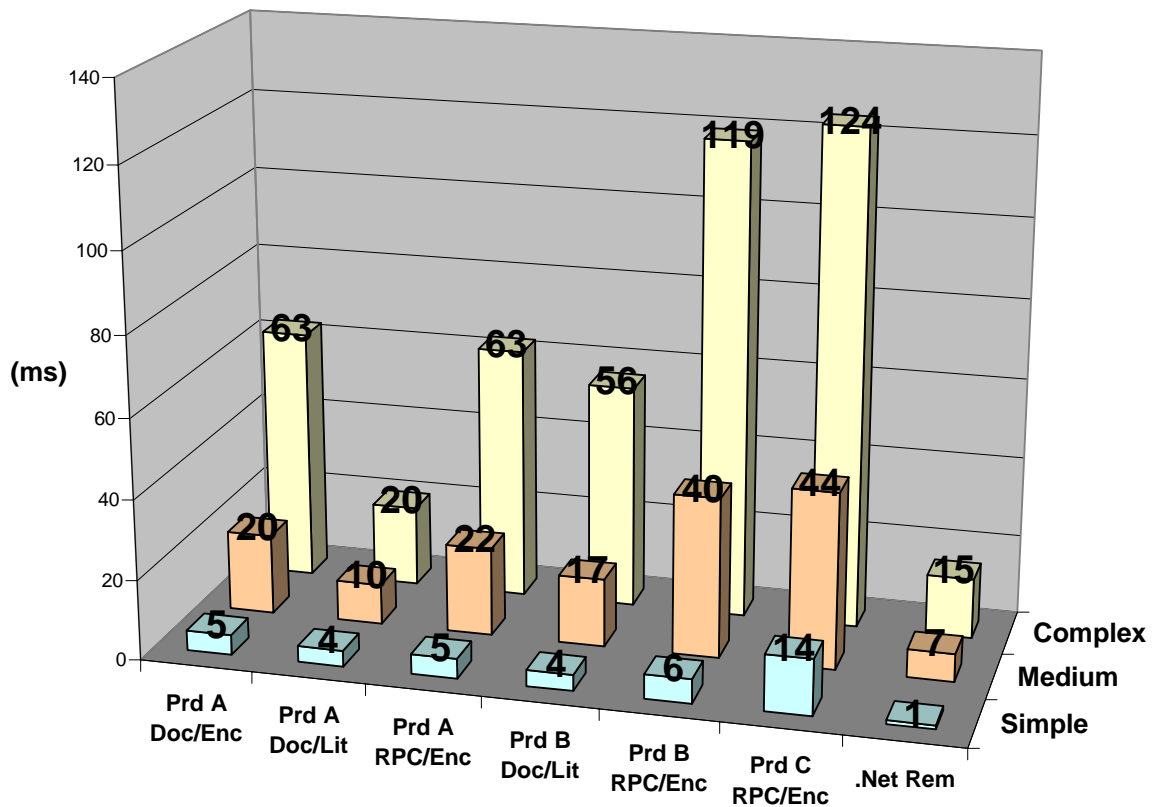


Figure 5. Message latency results

4.2.3 Complex Message Latency

The results of the complex message latency tests continue the trends seen in the simple and medium tests, as the longer messages require more serialisation and deserialisation time (made worse by the use of the SOAP encoding style), and more network transfer time. The leaders in this test are still the two Document/Literal implementations with a significant 36mS gap opening up between Product A and Product B. The binary/TCP .NET Remoting comes in at 15mS, only slightly faster than Product A's SOAP over HTTP Document/Literal implementation at 20mS but significantly faster than all the other SOAP combinations and implementations.

4.3 Throughput Analysis

We repeated the tests described in the previous section using a number of concurrent threads to measure server throughput: the maximum number of requests that a server can handle in a set time period. The number of test driver threads was steadily increased to ensure that the servers reached saturation. The results of these tests are reported in messages per second (msgs/sec).

4.3.1 Simple Message Type Throughput

Figure 6 shows the results obtained from the simple messages throughput test. These range from a low of 129 msgs/sec for Product C's implementation of RPC/Encoded up to 451 msgs/sec for Product A's implementation of the Document/Literal encoding style. Most RPC/Encoded implementations showed a throughput level of 350 msgs/sec while the two Document/Literal implementations offered higher throughput ranging from 390 to 450 msgs/sec. .NET Remoting gives a throughput number of about 1200 msgs/sec, significantly above the best of the SOAP/HTTP encoding styles and implementations, and an order of magnitude higher than the lowest.

4.3.2 Medium Message Type Throughput

Figure 7 shows the throughput results obtained from sending and receiving messages of medium size and complexity. The trend of these throughput results follows that obtained for the simple messages, but the results show lower overall numbers due to the larger messages, and subsequent higher serialisation and deserialisation costs. The throughput exhibited by the various products and style/encoding options has started to spread more widely in this test, with the leader

(Product A Document/Literal) giving about 5 times the throughput of the poorest performers (Products B and C with RPC/Encoded). .NET Remoting at 505 msgs/sec is still about 3 times higher than the best of the SOAP implementations.

4.3.3 Complex Message Type Throughput

Figure 8 shows the throughput performance for complex messages. We found that the gap between SOAP and non-SOAP implementations continued to widen with .NET Remoting offering 280 msgs/sec at peak while most SOAP implementations were only handling from 30 to 60 msgs/sec. Even the leading Product A Document/Literal implementation only gave a maximum throughput of 67 msgs/sec. The two lowest performing RPC/Encoded implementations only handled 15 msgs/sec, almost 20 times lower than the binary/TCP alternative.

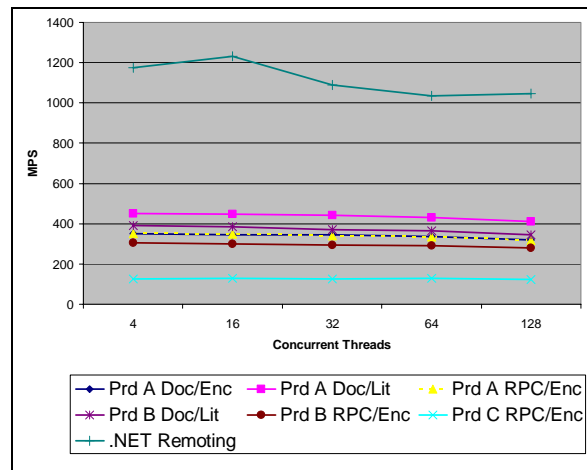


Figure 6. Throughput for simple messages

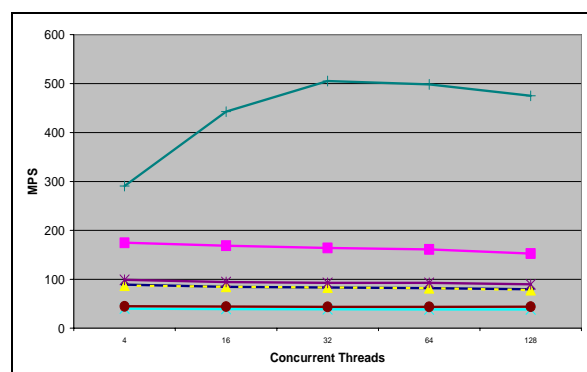


Figure 7. Throughput for medium messages

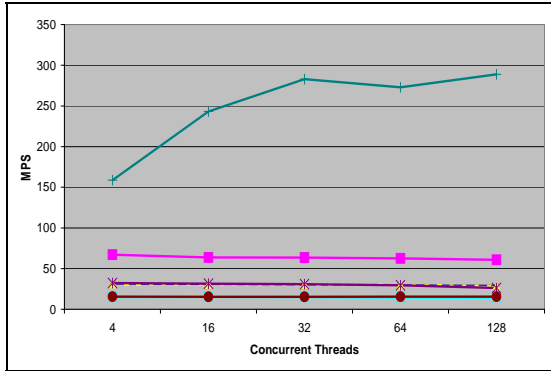


Figure 8. Throughput for complex messages

4.4 Serialisation and Deserialisation Analysis

A separate exercise was undertaken to measure SOAP serialisation and deserialisation overheads using the [TraceExtension()] method supported by Microsoft's ASP.NET. The test driver sent a number of customer detail records to the server and we captured the times required by the server to perform serialisation and deserialisation. Our implementation of the [TraceExtension()] method added an average of 260mS delay to the observed latency, due to overheads

in using system monitor counters. We consider the figures obtained from the experiments for large message size are still able to give us accurate insights into the costs of XML serialisation and deserialisation in SOAP.

A breakdown of the costs of sending 5000 customer detail records using both Document/Literal and RPC/Encoding implementations is shown in Figure 9. We observed that in both cases, the deserialisation overheads for processing incoming messages are higher than the serialisation overheads. Documents/Literal implementation takes 23.8% of the total time for deserialisation and 13% for serialisation, while RPC/Encoded implementation takes 30.7% of total time for deserialisation and only 10.7% for serialisation. The time taken to perform serialisation and deserialisation for the Document/Literal implementation are faster than the RPC/Encoded case by 100%, and 200% respectively. This experiment confirms that serialisation and deserialisation are the bottlenecks in both Document/Literal and RPC/Encoded implementations, with deserialisation overhead higher than that of serialisation.

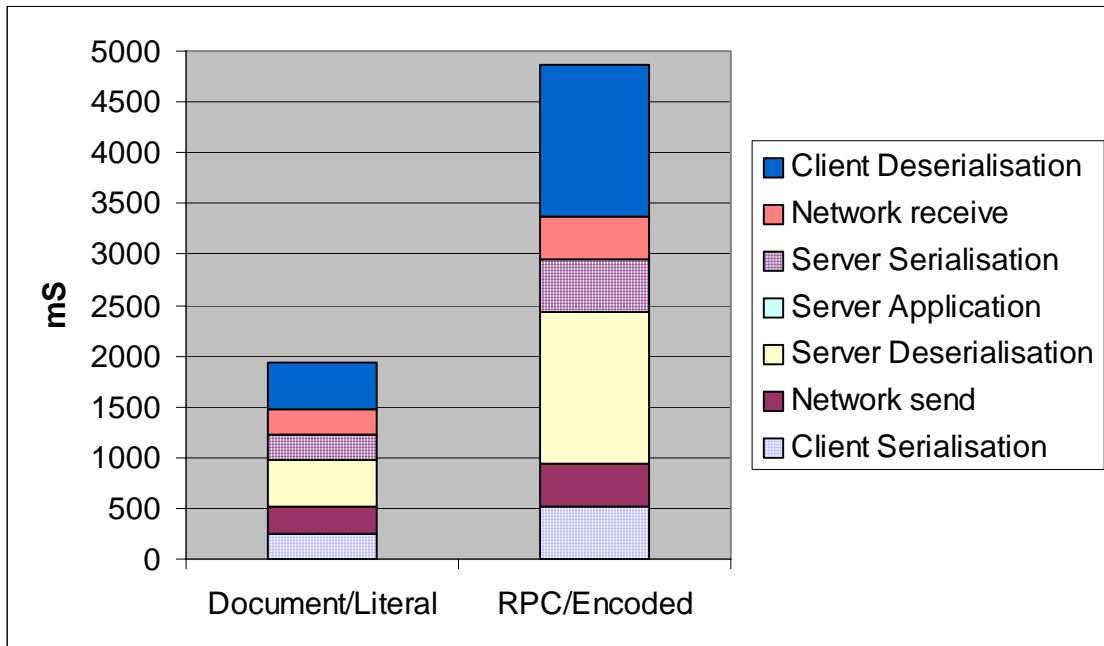


Figure 9. SOAP performance factors

5. Related Work

Earlier authors have looked at SOAP performance and compared: (a) SOAP and other middleware technologies [4, 5, 6, 8]; (b) SOAP and data exchange protocols [9], and (c) a number of SOAP implementations [3, 5]. It has been noted that SOAP is slower than existing middleware technologies [5, 6], other communication protocols [4] and other data exchange protocols [9].

The published studies show some interesting results, including that serialisation and deserialisation are the primary bottlenecks, and make some good suggestions about improving performance [4]. Some of the reported performance figures come from private SOAP implementations [4, 8]. Some of the evaluations are only interested in specific technologies, for example [3] compares the performance of JAXM implementations against JAXRPC, and [6] compares the performance of .NET Remoting against ASP.NET. Our evaluation differs from these studies in the way that we define our test scenarios to reflect commercial usage, and our use of commercial implementations from leading industry vendors.

6. Conclusion

We have successfully undertaken benchmark studies for some widely used, commercially available, SOAP implementations using workloads that reflect a range of typical commercial activities.

Our results show that the majority of current SOAP implementations are able to deliver reasonably good performance when handling short messages, with the best implementations coming close to binary/TCP performance. However, when the size and complexity of the messages increases, the performance of SOAP implementations gets comparatively worse, especially for the RPC/Encoded implementations. It is also interesting to note that although both the Document/Literal implementations perform better than any of the RPC/Encoded implementations, there is still a sizable gap between the performance shown by the two Document/Literal implementations, with Product A being the clear leader in all tests. We also confirmed that the processes of serialisation and deserialisation are the primary bottlenecks when processing large SOAP messages and that the overhead of deserialisation is higher than that of serialisation. These findings only apply for the case of SOAP running over HTTP transport across a local switched

LAN, and further tests are underway to look at the additional performance factors that come into play when SOAP over HTTP is used across wide-area networks.

References

- [1] Austin, D., Barbir, A., Ferris, C., et al. *Web Services Architecture Requirements*, W3C Working Group Note 11 February 2004
- [2] Ballinger, K., Ehnebuske, D., Gudgin, M., et al. *Basic Profile Version 1.0a Final Specification*, 2003/08/08 17:00:01 (on-line) Accessed Date: 19 August 2003 <http://www.wsi.org/Profiles/Basic/200308/BasicProfile-1.0a.html>
- [3] Benkner, S., Brandic, I., Dimitrov, A., et al. Performance of Java Web Services Implementations. In *Proceedings of ICWS'03*. Las Vegas, 2003
- [4] Chiu, K., Govindaraju, M., and Bramley, R. Investigating the Limits of SOAP Performance for Scientific Computing. In *Proceedings of 11th. IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02)*. Edinburgh, Scotland, p. 246-254:IEEE, 2002
- [5] Davis, D. and Parashar, M. Latency Performance of SOAP Implementations. In *Proceedings of IEEE Cluster Computing and the GRID 2002 (CCGRID'02)*. Berlin, Germany, IEEE, 2002
- [6] Dhawan, P. *Performance Comparison: Exposing Existing Code as a Web Service*, October 2001 (on-line) Accessed Date: 10 October 2002 <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbdta/html/bdadotnetarch11.asp>
- [7] Donald F. Ferguson, Brad Lovering, Tony Storey and John Shewchuk. Secure, Reliable, Transacted Web Services: Architecture and Composition. *MSDN technical article*, Sept. 2003. www.msdn.microsoft.com
- [8] Govindaraju, M., A. Slominski, et al. Requirements for and Evaluation of RMI Protocols for Scientific Computing. *Supercomputing*. IEEE, 2000
- [9] Kohlhoff C. and Steele R. Evaluating SOAP of High Performance Business Applications: Real-Time Trading Systems. In *Proceedings of WWW2003*, Budapest Hungry, 2003 www.uts.edu.au
- [10] Mitra, N. *SOAP Version 1.2 Part 0: Primer*, W3C Recommendation 24 June 2003, W3C. <http://www.w3.org/TR/soap12-part0/>
- [11] Fell S. tcptrace, PocketSOAP <http://www.pocketsoap.com/tcptrace/>