

Performance Considerations for Mobile Web Services

M. Tian, T. Voigt, T. Naumowicz, H. Ritter, J. Schiller

Freie Universität Berlin, Germany

Institut für Informatik

{tian, voigt, naumowic, hritter, schiller}@inf.fu-berlin.de

Abstract

Web services are an emerging technology that provides a flexible platform for web interaction. We evaluate Web service performance of handheld resource-constrained clients using different wireless technologies. Due to the usage of XML, message sizes in Web services are larger than in traditional web technologies and therefore, compression of Web service messages is attractive; as our experiments show in particular for mobile clients with poor connectivity and high communication costs. However, compression requires CPU time at both the server and the clients. We present measurement results of a simple dynamic scheme that provides benefits by compressing responses only when the required server resources are available.

1 Introduction

Traditionally, Internet servers such as web servers served mainly static content. During recent years, more and more web sites have started serving dynamic content, thus enabling personalization of web pages as well as more complex interaction such as on-line commerce and electronic banking. The latest trends in the field of web interaction are *Web services*. Web services are software components that can be accessed over the Internet using popular Web mechanisms and protocols such as HTTP. Public interfaces of Web services are defined and described using Extensible Markup Language (XML) based definitions. Examples of Web services range from simple requests such as stock quotes or user authentication to more complex tasks such as comparing and purchasing items over the Internet.

In contrast to traditional web interaction, Web services incorporate some additional overhead. In particular, due to usage of XML, requests and replies are larger compared to traditional web interactions and the need for parsing the XML code in the requests adds additional server overhead. We present a typical web application that requires the transmission of four to five times more bytes if implemented as a Web service compared to the same service implemented as a traditional dynamic program (in our case as an Active Server Page application). Therefore, compression of Web service interactions is attractive. It is easy to imagine that in the future clients using mobile devices will generate a large percentage of all Web service requests. Although the computing power of handheld devices is increasing rapidly the CPU time required for decompression might eliminate the benefits of compression for these types of devices. We present experiments that quantify the decompression overhead on a handheld computing device with constraint processing capabilities. As expected, mobile clients benefit from compression when the available bandwidth is scarce, for example when the client is connected via GPRS. But even when resource-constrained devices have better connectivity, the performance loss caused by decompression is almost negligible. Note that mobile clients also might prefer compressed responses since they are often charged by volume rather than by connection time, e.g. in the case of GPRS [15].

A lightly loaded server can afford the extra cost of compressing responses. We present measurements that show that the throughput of a heavily loaded server can decrease substantially when it is required to compress Web service responses. At the same time the response times experienced by the clients increase. We propose a simple scheme that allows clients to specify whether they want to receive data compressed when requesting a Web service. Depending on the current server load, the server compresses only the requests of the clients that required such a service. We present experiments that demonstrate that this approach works as expected and that both the server and clients with poor connectivity benefit during high server demand.

The main contributions of this paper are the evaluation of Web service performance for mobile clients as well as a scheme that supports a server in the decision whether to compress Web service responses.

The rest of the paper is outlined as follows: Section 2 presents some background information on Web services and the associated overhead. Section 3 motivates and discusses our dynamic compression approach. The following section presents our experiments. After presenting related work in Section 5, we conclude with a short summary of our findings and discuss some future work.

2 Web Services

A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML [1]. The definition of a Web service can be exported to a file, published to a lookup service, and discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

The Web service architecture defined by the W3C enables application to application communication over the Internet. Web services allow access to software components through standard Web technologies, regardless of platforms, implementation languages, etc.

In term of the Internet reference model, the Web service layer could be placed between the Transport and Application Layer. The Web service layer itself is based on several standard Internet protocols, whereby the lowest three of the protocols (WSDL, SOAP, and typically HTTP) as depicted in Figure 1 should be supported by all Web service implementations for interoperability.

The HTTP protocol that builds the first layer of the interoperable part of the protocol stack is, because of its ubiquity, the de facto transport protocol for Web services. But any other transport protocols such as SMTP, MIME, and FTP for public domains as well as CORBA and Message Queuing protocols for private domains could be used instead.

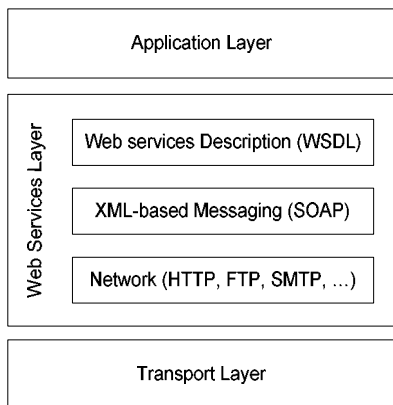


Figure 1: Web service protocol stack

The XML-based SOAP forms the next layer. SOAP provides XML-based messaging. In combination with HTTP, XML function calls can be sent as payload of HTTP POST. Because of the extensibility of SOAP, one can define one's own messages using SOAP headers. The highest interoperable layer is the XML-based Web Services Description Language (WSDL). A WSDL document serves as a contract to be followed by Web service clients. It defines the public interfaces and mechanisms of Web service interactions.

2.1 Web Service Overhead

Since both SOAP and WSDL are XML-based, XML messages have to be parsed on both the server and the client side and proxies have to be generated on the client side before any communication can take place. The XML parsing at runtime requires additional processing time, which may result in longer response time of the server in case of a Web service server.

In order to demonstrate the quantity of the additional bytes Web services generate for transfer, we have implemented the same "service" both as a traditional dynamic program, in our case as an Active Server Page (ASP) application, and as a Web service. The implemented application is an electronic book inventory system. The clients send the ISBN of a book to the server and the server returns information about the book such as the title, author name, price, and so on.

When sending small amounts of content using SOAP on HTTP, such as sending an ISBN for querying book information, the major part of the entire conversation will consist of HTTP headers, SOAP headers including the XML schema as well as brackets. In our case, the Web service accepts the ISBN of a book as input parameter and returns the book information in form of a dataset. The actual content of both request and response consists of a total of 589 bytes, thereof 10 bytes for the ISBN and the rest for the information about the book. But more than 3900 bytes have to be sent and received for the entire conversation. Figure 2 depicts the bytes on the wire for the actual content and the overhead when it is transmitted as HTML or XML. The disproportion is not as big for traditional web interaction with HTML. The total amount of the request and response for transferring the same information value is about 1200 bytes.

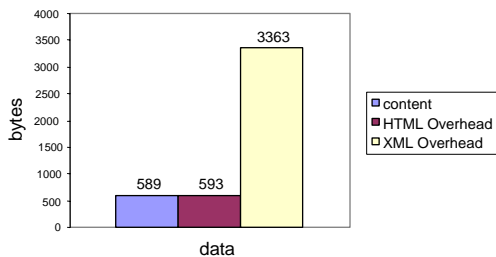


Figure 2: Overhead of server page and Web service

The overhead of the Web service stems mainly from the usage of XML producing human readable text and is employed when interoperability with other Web services and applications is essential [3]. Others have compared XML's way of representing data with binary encodings. They quantify the overhead as 400% [11].

3 A Dynamic Approach for Reduction of Web Service Responses

The growth of the Web service message size, which results in higher data transmission time, creates a critical problem for delay sensitive applications. One way to achieve a compact and efficient representation is to compress XML – especially when the CPU overhead required for compression is less than the network latency [11]. Compression is both useful for clients that are poorly connected as well as for clients that are charged by volume and not by connection time by their providers. The latter group contains mobile users connected with handheld devices such as people accessing a service via GPRS. This group of users is expected to increase rapidly in the next years. However, the Web service application on the server does not have any information about the delay, for example the current round trip time estimated by TCP, and about the available bandwidth between client and server.

Thus, we have decided to let the Web service users specify whether they want the response compressed. Mobile users usually know if they are charged by volume and often know how they are connected. Thus, it seems reasonable to let them decide whether they want the server to compress the response. In our current design we let users (or the clients' software) decide between three options:

- Do not compress the response
- Compress the response
- Compress the response if possible

If users choose the last option, the server is free to choose what the server considers best. To give users an incentive to choose this option, commercial Web service providers could decide to charge a lower price for this option. The choice of the users is reflected in the request. When the last option is chosen and the server demand is low, the server compresses the responses to all clients that have asked for compressed replies and to those clients that have not specified a preference. During high server demand, the server compresses only responses to clients that have asked for compressed data. Since compression requires mainly CPU time, we regard the server demand as high when the CPU utilization of the server exceeds a certain threshold.

Note that in this approach, the server can still become overloaded. Mechanisms for server overload protection have been studied elsewhere [7].

4 Experiments

In this section we describe our experimental setup and the application we have implemented as well as our experiments and the corresponding results.

4.1 Testbed

Our testbed consists of three 1GHz Pentium III machines with 256 MB memory, a Pentium III laptop with 700MHz and 384 MB RAM and an iPAQ Pocket PC 3970 running Windows CE 3.0 with a 400MHz XScale Processor (see Figure 3). Our Internet server is a standard Internet Information Server version 5.0 with the default configuration. The other two Pentiums run Linux. One is running the sclient traffic generator (see below) and the other runs NIST Net [10]. NIST Net emulates a wide

variety of network conditions such as low bandwidth and large round trip times. The iPAQ handheld device is connected to the server via the laptop and the machine running NIST Net.

For background load generation, we use the sclient traffic generator [2]. Sclient is able to generate client request rates that exceed the capacity of the Internet server. This is done by aborting requests that do not establish a connection to the server in a specified amount of time. This timeout is set to 50 milliseconds in our experiments. The exact timeout value does not impact the results, as long as it is chosen small enough to avoid that TCP SYN's dropped by the server are retransmitted. However, the larger the value, the higher the risk that the request generator runs out of socket buffers. Sclient does not take the aborted requests into account when calculating the average response time.

In order to emulate poorly connected clients we use the host running NISTNet to add additional delays and decrease the available bandwidth. We emulate a GPRS network based on the results from measurements in a real GPRS network by Chakravorty et al. [14]. They measure the delay on the uplink to about 500 ms and on the downlink to about 800ms. We do not take the variations into account in our experiments. For the bandwidth the theoretical values are 40.2 kbit/s on the downlink and 13.4 kbit/s on the uplink meaning that the mobile device listens simultaneously on three downlink channels while sending on one uplink channel as many mobile telephones do. The values measured by Chakravorty vary substantially based on the current network conditions with the best conditions coming very close to the maximum values. We use both the theoretical values as the best case from the clients' point of view as well as the values they measured when link conditions were poor. The latter are 12.8 kbit/s on the downlink and 4 kbit/s on the uplink.

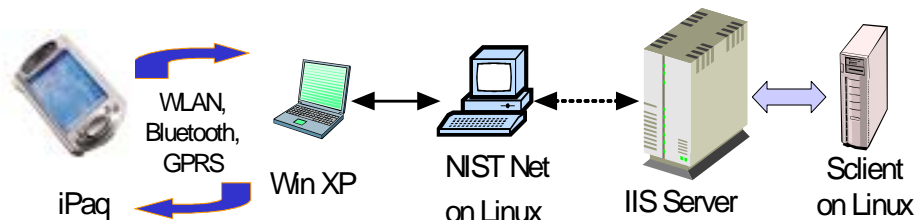


Figure 3: Testbed

4.2 Test Application

The implemented application is a modification of our electronic book inventory system described in Section 2.1 that returns responses of different sizes depending on the request. The additional requests are for a small Hello World service, and more detailed (“heavy”) information (including more detailed information, user ratings and hints on similar books etc.) about one, two, three and five books. The corresponding sizes of the SOAP body in both compressed and uncompressed form are shown in Table 1. Note that additional bytes are needed for the SOAP header (approx. 150 Bytes), the SOAP envelope (approx. 200 Bytes) and the HTTP header. We only compress the SOAP body in our experiments. We see that except for the small response the compression factor is about three.

The Web service running on a Microsoft Internet Information Server is implemented with the .NET framework 1.1 beta [13]. The Web service client is implemented with the .NET compact framework and is deployed on the iPAQ. Since SOAP is used for the client server interaction, we have extended the SOAP headers with the SOAPEExtension class of the .NET framework class library in order to modify (on the client side) and inspect (on the server side) SOAP messages. The client sets a parameter instructing the server either to compress or not to compress the data part of the SOAP response or to let the server decide by itself. For compression we use the SharpZipLib library [8], but other compression algorithms may be used.

As we discuss in Section 6, WSDL is a better place for such information but for ease of implementation we have implemented it in the SOAP header.

	Original data size (byte)	Data size after compression (byte)	Compression time on server (ms)	Decompression time on client (ms)
"Hello world!" response	209	256	1	41
Lite information for 1 book	3366	1390	12	200
Heavy information for 1 book	16055	6038	24	497
Heavy information for 2 books	28153	10222	36	747
Heavy information for 3 books	36049	12350	79	877
Heavy information for 5 books	50205	15470	89	1122

Table 1: Response size without and with compression, compression and decompression time

4.3 Experimental Results

In this section, we present three different set of experiments. In the first subsection we evaluate the performance of Web services for different wireless networks. These experiments show that mobile clients can gain from compression when their connectivity is poor. However, compression requires server resources and, therefore, we quantify the server overhead for compression. In the second subsection we demonstrate that compression can degrade server performance severely. The experiments in the final subsection validate our dynamic approach, where during high server load the server compresses only responses for clients that have indicated that they want the server to compress the response.

Web Service Performance for Handheld Devices

Due to the large message sizes of Web services we assume that compressing Web service responses is useful. However, the cost of decompression on resource-constrained devices may invalidate this assumption.

In the following we investigate the performance of Web Services on handheld devices when connected to different networks, namely with 802.11b wireless LAN, Bluetooth as well as GPRS networks with both good and poor connectivity, as

described in 4.1. The server in this scenario is lightly loaded because the client is the single user. Table 1 shows the compression times on the server and the decompression times on the client for different data sizes. The results indicate that the compression time on the server is much lower than the decompression time on the resource-constrained handheld device. On the iPAQ, the decompression time is more than one second for the largest response while compressing on the server requires less than 90 ms.

Figure 4 to Figure 7 show the experimental results with different network connections. The x-axis shows the original message size of Web service responses ranging from 0 to 50000 bytes. The y-axis is the service time in millisecond. The service time denotes the time interval between the moment the client requests the service, e.g. by clicking on a button and the moment the client has received and processed the result. We expect that mobile clients will benefit from compression when the bandwidth is scarce but experience small performance degradation when the available bandwidth is larger, i.e. when the service is requested over the wireless LAN or Bluetooth.

Since the available bandwidth in a wireless LAN is higher than in a Bluetooth network, we expect the service time in the wireless LAN to be lower than the service time over Bluetooth. Indeed, as shown in Figure 4 and Figure 5 the service time in the wireless LAN scenario is about 2 seconds faster than in the Bluetooth scenario for all message sizes. There is no significant difference between service time for compression and no compression when the iPAQ is connected via a wireless LAN or Bluetooth network. At its maximum, the time difference for receiving compressed or non-compressed responses is about 4% for the largest request over the wireless LAN. This shows that the overhead caused by compression is not severe in these scenarios.

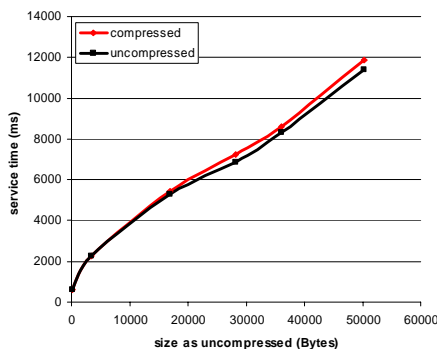


Figure 4: iPAQ service time over wireless LAN

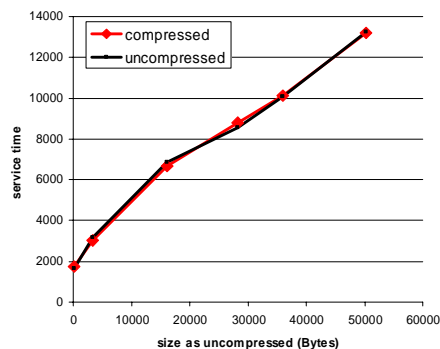


Figure 5: iPAQ service time over Bluetooth

When the iPAQ is connected via a low bandwidth network such as GPRS, the service time is lower for larger response sizes when the response is compressed. This means that the benefit of compression is higher than the cost of decompressing the response. As Figure 7 shows, when connectivity is poor the service time is halved when compressing the largest response.

These experiments demonstrate that compressing Web service responses is useful when the available bandwidth is scarce even for clients using resource-constrained devices.

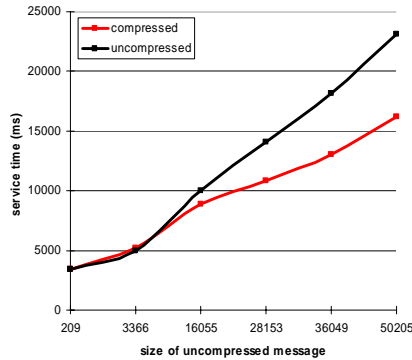


Figure 6: iPAQ service time over emulated GPRS network

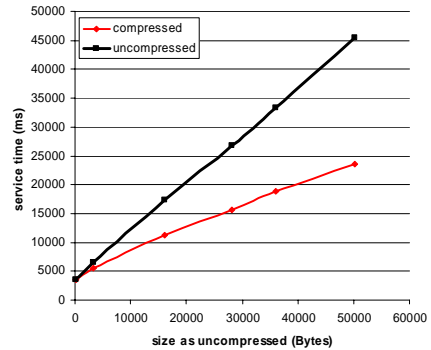


Figure 7: iPAQ service time over emulated GPRS with poor connectivity

Impact of Compression on Server Performance

Compression requires also CPU time at the server. In this section, we evaluate the impact of compression on server performance. We use the sclient workload generator to sustain a certain request rate independent of the load on the server. The traffic generator makes requests at a certain rate for the electronic book Web service described in the previous section.

The test scenario increases the request rate across runs and conducts three runs for each request rate with each of the runs lasting for three minutes. We measure the average throughput and response time. We expect that the response time will be quite low when the request rate is below the capacity of the server, no matter whether compression is used or not. However, using compression we expect that the server will reach its maximum capacity at a lower request rate. When the request rate is above the capacity of the server, the response time will increase rapidly due to the waiting time that requests spend queuing before they can be processed. Also, the throughput will increase with the request rate until the maximum server capacity is reached. When the request rate is higher than the capacity of the server, the throughput will not increase anymore. During severe overload the throughput might even decrease since CPU time is wasted on requests that cannot be processed and are eventually discarded [2].

For compression we use the SharpZipLib library. This way, we can reduce the overall number of bytes for the “lite information” scenario from more than four KBytes to around two Kbytes (cf. Table 1). Without compression, three TCP segments are needed for the response while the compressed response fits into two TCP segments. Note that in our experiments the client, i.e. the traffic generator, is not required to decompress or to process the received data in some other way. Due to the additional CPU time the server spends on compressing data, we assume that the response time increases and the throughput (measured in connections per second) decreases when the response is compressed. Figure 8 shows that this is indeed the case. The response time during overload is about three seconds higher and the

throughput is about 45 conn/sec lower when compression is used. Figure 9 shows that the maximum server throughput decreases from about 135 conn/sec to 90 conn/sec when compression is used. These experiments show that when a server compresses all replies, the maximum server throughput decreases substantially and the response time experienced by the clients is affected negatively. This gives reason for our approach that is based on the assumption that servers should only compress replies to clients that can benefit from compression.

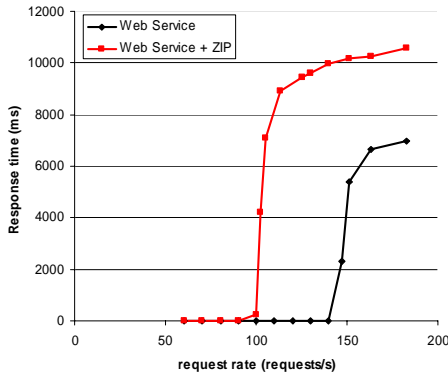


Figure 8: Comparison of response time with and without compressing the response

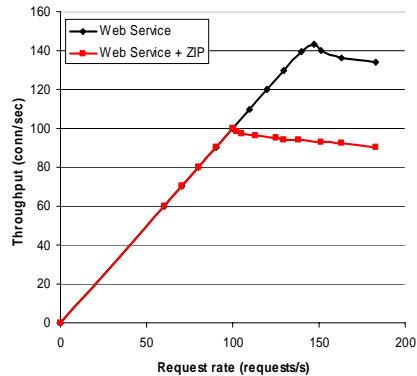


Figure 9: Comparison of throughput with and without compressing the response

Dynamic Server Compression

The first experiments have shown that mobile clients with poor connectivity benefit from compression while the experiments in the previous section have demonstrated that compression reduces server performance during high demand. In the experiment in this section, we investigate if the dynamic compression approach described in Section 3 is able to give us the best of both worlds, i.e. compressed data for clients that wish to receive compressed data while achieving high server throughput.

The next experiment compares the server performance when

- i) all responses are compressed,
- ii) no responses are compressed and
- iii) the server decides which responses to compress.

As in the experiment in previous section, we use sclient to request the lite information on a book. Sclient first requests that all responses are to be compressed, then that no responses are compressed and finally 50% of the responses to be uncompressed and for the other 50%, the server is asked to decide. In the latter setting, which we call dynamic, when no compression indication is given for a request, the server compresses the response when the CPU utilization of the server is below a threshold of 80%. If the CPU utilization is higher than 80% it does not compress such a response in order to save processing time. Using this approach, the performance should be almost as high as without compression. Figure 10 and Figure 11 show the response time and the throughput, respectively. As expected, using the dynamic approach the server performance is almost as high as when the server does

not use compression. Further inspection of the results reveals that the server compresses all responses it is allowed to compress (50% of the requests that have indicated that they do not have any preferences) until a request rate of 60 requests/s is reached. When the request rate reaches 120 requests/s only 20% of the requests without preference indication are compressed while no requests are compressed at request rates larger than 140 requests/s.

However, the performance gap should be smaller, i.e. the difference between the dynamic approach and no compression should be almost nothing, since the only extra task required from the server is to check the current CPU utilization when processing a request that has not indicated any preference. This indicates that this task is more expensive than one would expect.

In the next experiments we want to validate that a client with a poor connection to the server may indeed benefit from this approach. In these experiments, the sclient varies the request rate and requests the "lite information for one book" for two different scenarios. In the first scenario sclient requests compressed responses while in the second scenario sclient requests 50% of the responses as compressed and 50% of the responses without compression preference. In both scenarios, the iPAQ requests the "heavy information for one book". The mobile client is connected via the emulated GPRS network to the server.

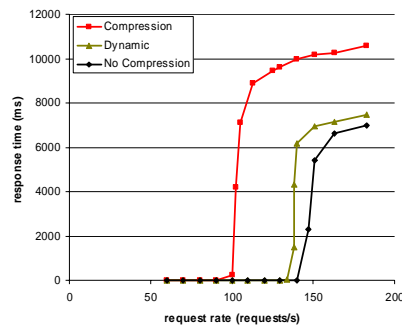


Figure 10: Response time of the dynamic approach

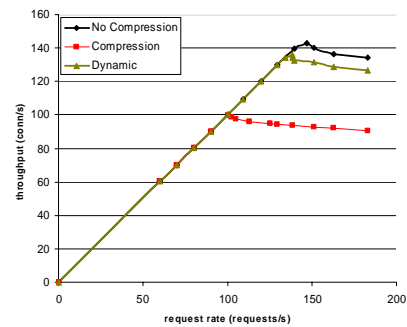


Figure 11: Throughput of the dynamic approach

The results shown in Table 2 indicate that our approach is beneficial for both the server and for mobile clients with poor connectivity. As expected for some sclient request rates, the service time experienced by the mobile client is much better when the server uses the dynamic approach, namely when the sclient request rates are between 100 and 140 requests/s which corresponds to the results in Figure 10. This is the request range where the CPU time required for processing would overload the server and would degrade performance but the server still performs well when it does not need to compress the responses.

Sclient rate (requests/s)	80	100	120	140	160
All compressed (ms)	9985	10495	16492	17104	17176
Dynamic (ms)	9370	9850	10468	10538	17131

Table 2: Impact of the dynamic approach on the response time of the mobile client

5 Related Work

Web Services are a young area of research, thus not much about Web services has been published yet. Cai et al. compare alternative encoding mechanisms, namely binary and XML, for Web services [3]. Their aim is to discuss the performance trade-off associated with these two alternatives. They develop a model that allows them to estimate the throughput depending on factors such as server bandwidth and packet loss. In our work we implement the described approach and demonstrate experimentally the benefits for both the server and poorly connected clients. Chiu et al. have investigated the limits of SOAP performance for scientific computing [5]. They describe an efficient SOAP implementation specifically targeted at systems with stringent memory and bandwidth requirements while we improve the performance of an existing implementation during high demand and particularly considering poorly connected clients.

Krashinsky investigates optimizing the final critical link between a mobile web client and a stationary base station by compressing HTTP request and reply [9]. While they use a proxy for compression over the last link we use end to end compression if desired by the clients. Ardon et al. present a generic content adaptation architecture using a distributed proxy architecture [16]; our focus is on compression for Web services which can be deployed exclusively on the server. While in our approach the clients themselves decide if they want compressed replies, Krishnamurthy and Willis classify client connectivity based on information collected during previous client access to the same web server [4]. In particular, they measure the delay between the base object and the first embedded object as well as the delay between the base object and the last object in a sequence. Ardaiz et al. measure the service time of web clients by extending web server logs to contain more precise information and compute the service time from the server logs [12]. Note that in both cases there is no information about a client available when she makes her first request to the web site.

In contrast to traditional web interaction, little is known about request size and file distributions of Web services. Furthermore, the performance issues of traditional web servers are well understood [6] and mechanisms to cope with server overload have been developed [7].

6 Conclusions and Future Work

Compression is one way of dealing with the problem of large message sizes of Web services. We show that compression is useful for poorly connected clients with resource-constrained devices despite the CPU time required for decompressing the responses. Compression also decreases server performance due to the additional CPU time required. In the approach presented in this paper, we let the clients decide whether they want their responses compressed. During low demand, the server compresses the responses for all clients that have asked for compressed responses as well as for clients that have not indicated any preference. During high server demand, only responses to clients that have asked for compressed response are compressed. Our experiments have shown that both the server and the clients, in particular clients that are poorly connected, benefit from this approach.

In the presented experiments, the client can instruct the server to compress the returned data. This information could be seen as some kind of quality of service

(QoS). We think that a general QoS support for Web services will play an important role for the success of this emerging technology. Providers of QoS aware Web services can specify QoS related statements on the services they offer, while the clients can specify QoS related statements on services they require. The statements made by both the service provider and service requestor should be able to be matched, adjusted, and controlled at runtime. In our experiments, we have used the SOAP headers for a single QoS attribute. But we think that the WSDL is a better place for further integration of QoS information. In an ongoing research project, we are examining QoS support for Web services in a general way.

References

- [1] W3C, Web services Activity, <http://www.w3.org/2002/ws/>
- [2] G. Banga and P. Druschel. Measuring the capacity of a web server. Usenix Symposium on Internet Technologies and Systems, December 1997.
- [3] M. Cai, S. Ghandeharizadeh, R. Schmidt and S. Song. A Comparison of Alternative Encoding Mechanisms for Web Services. 13th International Conference on Database and Expert Systems Applications, Aix en Provence, France, September 2002.
- [4] B. Krishnamurthy and C. Wills. Improving Web experience by client characterization driven server adaptation. Proceedings of WWW 2002 Conference, Hawaii, May 2002.
- [5] K. Chiu, M. Govindaraju and R. Bramley. Investigating the Limits of SOAP Performance for Scientific Computing. IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 2002.
- [6] E. Nahum, T. Barzilai, and D. Kandlur. Performance Issues in WWW servers. IEEE/ACM Transactions on Networking, Vol. 10, No. 1, February 2002.
- [7] T. Voigt, R. Tewari, D. Freimuth and A. Mehra. Kernel Mechanisms for Service Differentiation in Overloaded Web Servers. 2001 Usenix Annual Technical Conference, Boston, MA, USA, June 2001.
- [8] SharpZipLib, <http://www.icsharpcode.net/OpenSource/SharpZipLib/default.asp>
- [9] R. Krashinsky. Efficient Web Browsing for Mobile Clients using HTTP Compression. Distributed Operating Systems term project, Massachusetts Institute of Technology, Dec. 2000.
- [10] NIST Net, National Institute of Standards and Technology, <http://snad.ncsl.nist.gov/itg/nistnet/>
- [11] A. Mani and A. Nagarajan. Understanding quality of service for Web services. <http://www-106.ibm.com/developerworks/library/ws-quality.html>, Jan. 2002.
- [12] O. Ardaiz, F. Freitag, L. Navarro. Estimating the Service Time of Web Clients using Server Logs. ACM SIGCOMM Workshop on Data Communication in Latin America and the Caribbean, San Jose, Costa Rica, SIGCOMM Latin America 2001.
- [13] .NET Framework class library, version 1.1.4322, November 15, 2002.
- [14] R. Chakravorty, J. Cartwright and I. Pratt. Practical Experience with TCP over GPRS. IEEE GLOBECOM 2002, Taipei, Taiwan, Nov. 2002.
- [15] J. Schiller. Mobile Communications. Addison-Wesley, 2000.
- [16] S. Ardon, P. Gunningberg, Y. Ismailov, B. Landfeldt, M. Portmann, A. Seneviratne and B. Thai. Mobile Aware Server Architecture: A distributed proxy architecture for content adaptation. INET 2001, Stockholm, Sweden, June 2001.