PERFORMANCE IMPACT OF WEB SERVICES ON INTERNET SERVERS

M. Tian, T. Voigt, T. Naumowicz, H. Ritter, J. Schiller
Freie Universität Berlin
Computer Systems & Telematics
{tian, voigt, naumowic, hritter, schiller}@inf.fu-berlin.de

Abstract
While traditional Internet servers mainly served static and later also dynamic content, the popularity of Web services is increasing rapidly. Web services incorporate additional overhead compared to traditional web interaction. This overhead increases the demand on Internet servers which is of particular importance when the request rate to the server is high. We conduct experiments that show that the imposed overhead of Web services is non-negligible during server overload. In our experiments the response time for Web services is more than 30% higher and the server throughput more than 25% lower compared to traditional web interaction using dynamically created HTML pages.

Key Words
Web computing, Web services, Internet server, performance, compression, XML overhead

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 enabling the personalization of web pages as well as more complex interaction such as on-line e-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. 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 that places more demand on Internet servers. In particular, due to the usage of XML, the requests and replies are larger compared to traditional web interactions and the need to parse the XML code in the request adds additional server overhead. During low demand, this may not cause any severe performance penalties, but when the server operates close to its maximum or the request rate exceeds the capacity of the server, this additional overhead may become non-negligible. While it is evident that Web services incorporate additional overhead, to the best of our knowledge, no studies have been conducted that have tried to quantify this overhead. Therefore, this paper tries to quantify the overhead from the perspective of a highly loaded Internet server.
In this paper we investigate and compare the behavior of Internet servers during overload for both traditional web interaction and Web services. Towards this end, we implement the same "service" both as a traditional dynamic program, in our case as an Active Server Page (ASP) application, and as a Web service. We call the former server page and the latter Web service. While in both cases the same “information value” is returned, about four to five times more bytes need to be transferred when the request is made using the Web service technology. This overhead stems mainly from the usage of XML producing human readable text and is employed when interoperability with other Web services and applications is essential [1].
When the request rate exceeds the capacity of the Internet server, i.e. the arrival rate of requests is higher than the service rate, the throughput decreases and the response time increases as expected. In our experiments, the server page can handle more than 25% more requests than the Web service while providing about 30% lower response times during moderate and high overload. When the bandwidth between the client and the server is scarce, for example, when mobile clients with low connectivity access the service, or the size of the response is large, compressing the Web service response might be useful. However, our experiments demonstrate that when the reply is compressed the response time increases and the server throughput decreases additionally due to the extra CPU power that is needed for compression.
Our findings show that when reimplementing traditional services as Web services, the additional overhead of the Web service must be taken into account when planning the capacity of the affected server.
The main contribution of this paper is the quantification of the additional overhead that Web services introduce to web interactions compared to traditional web interaction.
The rest of the paper is outlined as follows: Section 2 presents some background information, mainly on Web services and the Internet Information Server. In Section 3, we present our experimental setup and demonstrate and discuss our results. After discussing some related work, we conclude with Section 5.

2. Background
In this section we describe the Web service architecture and the Microsoft Internet Information Server (IIS). We

deploy the IIS with .NET Web service extension to host the server applications.

2.1 Web Services

A Web service is − by definition [2] − a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. The definition of a Web service can be exported to a definition file, published to a lookup service, and discovered by other software systems. These systems may then interact with the Web service in a manner described 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 platform, implementation languages, etc.
In terms of the Internet 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, called protocol stack, where the lowest three of the protocols depicted in Figure 1 should be supported by all Web service implementations for interoperability.

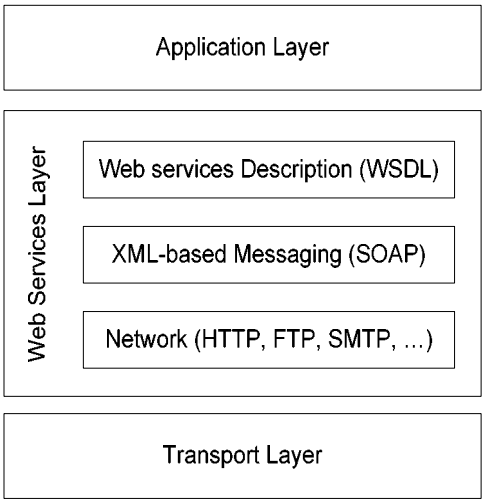

Figure 1: Web service protocol stack

HTTP building 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 domain as well as CORBA and Message Queuing protocols for private domains could be used instead.
The XML-based SOAP forms the next layer. SOAP ensures 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 customized messages using SOAP headers.
The highest interoperable layer is the XML-based Web Service Definition Language (WSDL). A WSDL document serves as a contract to be followed by Web service clients. It defines the public interfaces and mechanics of Web service interactions.
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 a longer response time of the server in case of a Web service server.

2.2 Internet Information Server (IIS)

We deploy Visual Studio .NET enterprise edition with ASP.NET supporting Web services on a Windows 2000 Server machine with IIS 5.0. The IIS intercepts all client requests on port 80 and passes them to the ASP.NET ISAPI filter. The filter connects to a worker process named aspnet_wp.exe (ASP.NET worker process), which implements the HTTP pipeline that ASP.NET uses to process web requests.
The ASP.NET worker process decides according to the request type whether the received HTTP packets belong to a SOAP request or a page request. Special HTTP handlers (PageHandlerFactory, WebServiceHandler-Factory) are then responsible for parsing and compiling the bytes of ASP.NET (.aspx) and Web service files (.asmx), and generating the response to the client. [3]

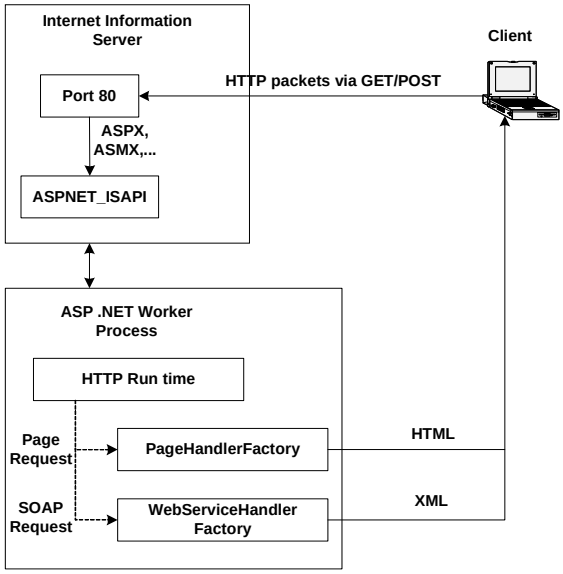

Figure 2: The ASP.NET architecture [3]

3. Experiments

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

3.1 Testbed

Our testbed consists of two 1GHz Pentium III machines with 256 MB memory that are directly connected to each other. The client machine used for traffic generation is running Linux and our server machine runs the Windows 2000 server edition. Our Internet server is a standard

Internet Information Server version 5.0 with the default configuration. Note that this setup does not allow us to assess the effect of network latency on the response times.
For load generation we use two different versions of the sclient traffic generator [4]. The first version which we call singleclient emulates a specified number of clients. Each client makes a request to the server and waits for the response. Immediately after receiving the response, it makes another request. Thus, singleclient can put a high load on the server when the number of emulated clients is high. Singleclient is similar to other "closed loop" tools such as webstone [5] that can put a high load on a server but are not able to characterise the behavior of a server during overload.
The second version which we call sclient in the following 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 low enough to avoid that TCP SYNs 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 into account aborted requests when calculating the average response time.

3.2 Test Application

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 title, author name, price, and so on. For comparison, we implemented the same application as a Web service and a server page (Active Server Page, ASP). Both applications return the same "information value". During the performance tests, the server does not perform any query to a data base or other websites; it just returns the prepared results after parsing the requests. We do not include any database operation in order to prevent the database from becoming the bottleneck and falsifying our results. SOAP messages are returned in the case of Web service requests (Figure 3) and HTML-based messages are returned when the server page is requested (Figure 4).

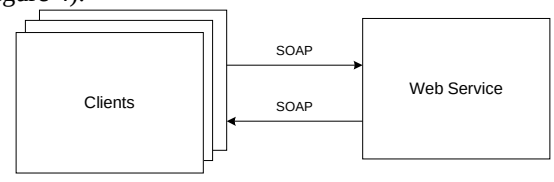

Figure 3: Interaction between clients and Web service

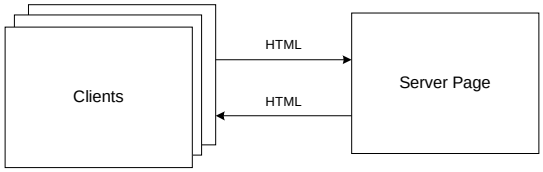

Figure 4: Interaction between clients and server page

Since SOAP is an XML-based messaging system and XML uses plain text for data representation, exchanging data via SOAP results in a size which is more than 500% larger than representing the same data in HTML format in our experiments (the HTML page is written as simple as possible).
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 protocols, 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 the request and response consists of a total of 589 bytes, including 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 5 depicts the bytes on the wire for the actual content and the overhead when it is transmitted over HTML and SOAP on HTTP.

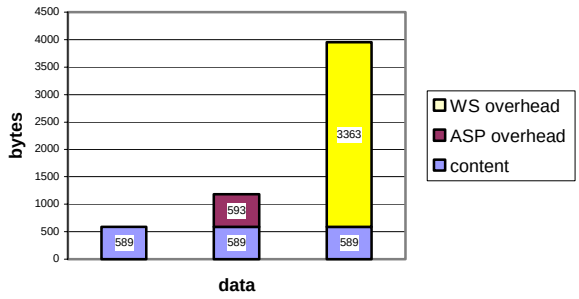

Figure 5: Overhead of server page and Web service

The disproportion is not as big for traditional web interaction and HTML. The total amount of the request and response for transferring the same information value is about 1200 bytes.

3.3 Experimental Results

3.3.1 Singleclient

In the first experiment, we use the singleclient workload generator to request the corresponding service from the server. For each run of the experiment, we configure singleclient with a static number of clients. We increase the number of clients from 1 to 500. We measure both the response time and the number of replies received per second by the clients. The latter denote the throughput of the server. It is worth noting that the clients do not process the contents of the responses.
We expect the response time to increase linearly with the number of clients, since we would expect each client to receive about 1/n of the server resources when n clients are active simultaneously. The throughput should increase with the number of clients until it saturates at the maximum capacity of the server in this experiment. The results are shown in Figure 6 and Figure 7. We expect the

maximum throughput to be higher and the response time for the server page to be lower than for the Web service.

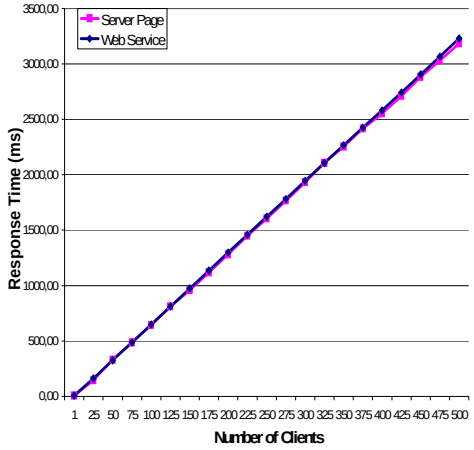

Figure 6: Response time with singleclient

Figure 6 shows that the response time measured by singleclient indeed increases linearly with the number of clients. Note that the difference between the server page and the Web service is not very high in this scenario.

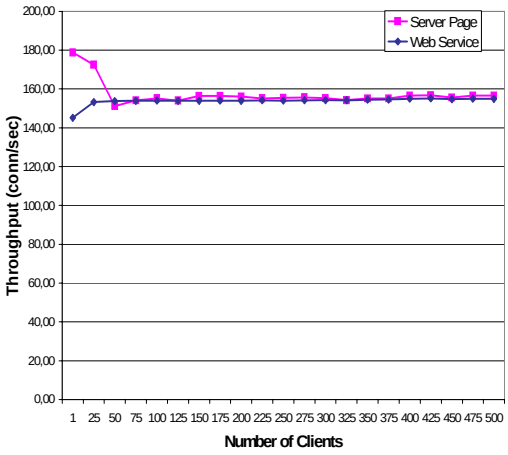

Figure 7: Throughput with singleclient

Figure 7 shows that the throughput measured by singleclient is about the same for the server page and the Web service when the number of clients is sufficiently high. However, for small client numbers, we can see that the server can handle more requests when the server page is requested. Both experiments seem to show that there is no significant difference in the performance between the Web service and the server page. The next experiments using sclient will show that this is not true when the request rate exceeds the capacity of the web server.

3.3.2 Sclient

In this experiment, we use the sclient workload generator that can sustain a certain request rate independent of the load on the Internet server. This time, we increase the request rate across runs. We conduct three runs for each request rate. Each of the runs has a duration of three minutes and we take the average. Since the standard deviation is very low, we only show the average in the graphs. Again, we measure the 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. When the request rate is above the capacity of the server, the response time will increase fast due to the waiting time that requests spend queued 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 that will eventually be discarded [4]. Due to the overhead of the Web service compared to the server page, we assume that the point where the response time increases and the throughput does not increase anymore occurs at a lower request rate for the Web service compared to the server page.

Figure 8 shows the response time achieved by sclient. For low request rates, the response times are low for both the server page and the Web service. As expected, the response time increases faster at a lower request rate for the Web service than for the server page, namely at about 145 request/s for the Web service and 180 request/s for the server page. After the fast increase, the response time increases only slightly for a certain range of request rates (150-200 for the Web service). The reason for this is that further connection requests are rejected. As mentioned in Section 3.1, we do not include rejected requests when computing the average response times. Therefore, the average response times increase only slightly for these request rates. For very high request rates a further increase of the response time is caused by the large number of interrupts that occur whenever a packet arrives at the server since interrupt processing has priority over application processing.

Figure 8: Response time with sclient

Figure 9 denotes the throughput for the server page and the Web service. For request rates up to 140, the server throughput is as high as the request rate (request rates 0 to 110 are omitted in the figure). While the throughput of the Web service decreases for larger request rates, the throughput for the server page reaches its maximum at a request rate of about 180 request/s and decreases again.

This decrease is caused by the CPU time that is wasted on requests that are later discarded as well as the larger number of interrupts due to packet arrivals.

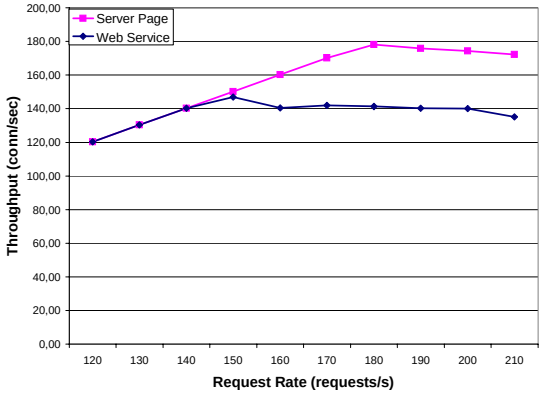

Figure 9: Throughput with sclient

As expected, the Web service reaches its maximum server throughput at a lower request rate than the server page. The reason for this is shown in Figure 10 that depicts the number of rejected connection requests. The figure shows that for the Web service connection requests are rejected at request rates larger than 150 request/s while the same is true for request rates larger than 190 request/s for the server page.

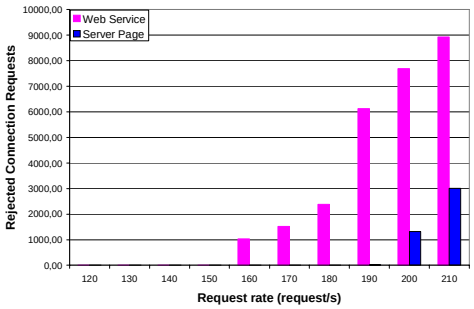

Figure 10: Rejected connection requests

3.3.3 Compression

One way of reducing the number of bytes transferred over the network is to compress the Web service response. This is useful when a large amount of data has to be transferred or the bandwidth between server and client is low. The latter is of particular importance for mobile users that have low connectivity and are often charged by volume of data sent and received. However, the extra CPU time required to compress the data on the server might be of importance during high server load. When clients have slow CPUs, the processing time required to decompress the data can also be high. In the next experiment we use the same setup and the same Web service as before, but require the server to compress the data before returning it to the client. We use the SharpZipLib library [6] for compression, but any other compression algorithms could be used. This way, we can reduce the number of bytes from more than three KBytes to 1.300 Bytes. Without compression, three TCP segments are needed for the reply while the compressed reply fits into one TCP segment. Note that in our experiments the client, i.e. the traffic generator, is not required to decompress the data. Due to the additional CPU time the server spends on compressing data, we assume that the response time increases and the throughput decreases when the reply is compressed.

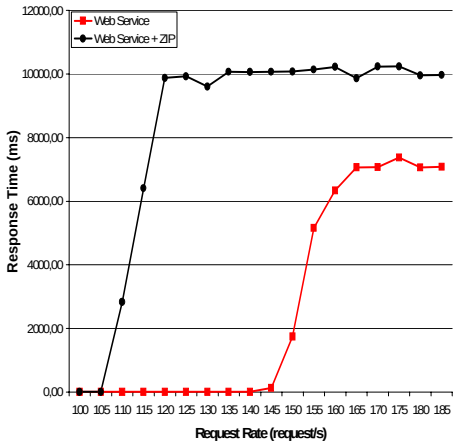

Figure 11: Comparison of Web service with and without compressing the reply

Figure 11 shows the case as expected. Although the response time during overload is about three seconds higher and the throughput is about 50 conn/sec lower, note that as discussed above compression might be useful in other scenarios. However, from the point of the server, compression requires extra CPU capacity. We assume that a simple solution to the problem is to let the server compress data until the CPU utilization reaches a certain threshold and stop compressing when the CPU utilization is higher than that threshold.

3.3.4 Summary of the Results

The results show that using closed loop tools does not enable us to capture the real behavior of the web interactions during overload and fails to evaluate the performance overhead of Web services compared with traditional web interaction. The results from the experiments with the sclient workload generator reveal that this overhead is non-negligible. Using our test application, the server page can handle more than 25% more requests than the Web service while providing about 30% lower response times during moderate and high overload.

4. 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 [1]. Their aim is to discuss the performance tradeoff associated with these two alternatives. They developed a model that allows them to

estimate the throughput depending on factors such as server bandwidth and packet loss. Our work also deals with server throughput, but we focus on the role of the Web service overhead during server overload. Chiu et al. have investigated the limits of SOAP performance for scientific computing [7]. They describe an efficient SOAP implementation specifically targeted at systems with stringent memory and bandwidth requirements while we evaluate the performance of a commercial server during overload. Menasce and Almeida present methods for capacity planning of Web services [8]. Their methods are general and do not consider the specific overhead of Web services compared to traditional web interaction.

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 [9] and mechanisms to cope with server overload have been developed [10].

5. Conclusions and Future Work

In this paper, we have compared the behavior of Internet servers during overload for both traditional web interaction and Web services. Web services impose additional overhead on the server since they require the server to parse the XML data in the request. Furthermore, the size of the transferred data increases. Compression can reduce this but requires additional CPU time. Our experiments have shown that the overhead imposed by Web services is non-negligible and can reduce server throughput and increase response times significantly during high server demand. We have used a single, however as we believe, typical test application and it would be interesting to look at a wider range of applications. Unfortunately, as stated in the previous section, little is known about the characteristics of Web services in form of request size distributions or processing requirements that would enable us to perform experiments that build on empirical foundations.

Apart from performance improvements for Web services, we plan to examine the performance impact of Web services on clients running on mobile devices. The usage of small and mobile devices is increasing rapidly. It is well known that mobile devices are resource constrained. They do not have as much CPU power, memory, hard disk, bandwidth, and so on as desktop computers. Therefore, it is useful to analyze the behavior of mobile Web services in order to improve performance and Quality of Service support for this increasing group of users.

References

[1] Cai, M. et. al., 2002, A Comparison of Alternative Encoding Mechanisms for Web services. 13th International Conference on Database and Expert Systems Applications, Aix en Provence, France.

[2] W3C, 2002, Web services Activity, http://www.w3.org/2002/ws/

[3] Esposito, D., 2002, Building Web Solution with ASP.NET and ADO.NET. Microsoft Press, Redmond, Washington, USA

[4] Banga, G. and Druschel, P., 1997, Measuring the capacity of a web server. Usenix Symposium on Internet Technologies and Systems

[5] Mindcraft, Webstone. http://www.mindcraft.com

[6] SharpZipLib, http://www.icsharpcode.net/OpenSource/SharpZipLib/default.asp

[7] Chiu, K. et. al., 2002, Investigating the Limits of SOAP Performance for Scientific Computing. IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland.

[8] Menasce, D. and Almeida, V., 2002, Capacity Planning for Web services, Prentice Hall, Upper Saddle River, NJ

[9] Nahum E. et. al., 2002. Performance Issues in WWW servers. IEEE/ACM Transactions on Networking, Vol. 10, No. 1.

[10] Voigt, T. and Gunningberg, P., 2002. Adaptive Resource-based Web Server Admission Control. IEEE Int. Symposium on Computers and Communication, Giardini Naxos, Italy.