Enhanced SOAP Performance for Low Bandwidth Environments

A thesis submitted in fulfilment of the requirements for
the degree of Masters of Applied Science

Khoi Anh Thi Phan
B.Eng.

School of Computer Science and Information Technology
Science, Engineering, and Technology Portfolio
RMIT University

August 2007

Declaration

I certify that except where due acknowledgement has been made, the work presented in this thesis is solely my original research. This work has not been submitted previously, in whole or in part, to qualify for any other academic award. The work has been carried out since the official commencement date of my candidature on 30th March 2005.

Khoi Anh Thi Phan
School of Computer Science and Information Technology
RMIT University
August 2007

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this research thesis. I would like to send special thanks to my two supervisors Professor Zahir Tari and Dr. Peter Bertok for their support and guidance during my candidature. Both Zahir and Peter have been great supervisors in motivating me on doing high quality research and keeping me on track on progress. Without their support, I would have not been able to come up with these ideas, all their comments and suggestions were always helpful.

I would also like to send my gratitude to Dr. Kwong Lai and Dr. Andrew Fry who have co-supervised me during the first semester and the final year respectively. Kwong's suggestions helped me to have interest in SOAP performance. He also guided me in writing my research proposal and understanding more about research during my first months at RMIT. Andrew has been an excellent adviser. His inputs in research directions and experimentation were invaluable. Especially, his proof-reading of my thesis is highly appreciated.

I am thankful to the School of Computer Science and Information Technology at RMIT for their financial support throughout my candidature. My sincere thanks go to the administration and research staff, Ms Beti Dimitrievska, Ms Nyree Koistinen and Dr. Michael Winikoff, in the School, because without their help I would not have completed this degree.

I am grateful to all the staff and fellow students in the Distributed Systems and Networking discipline for creating a good, productive and friendly environment for me to research and share knowledge with others. Especially, my sincere thanks go to friends in Room 10.9.22, Nalaka Gooneratne, Sakib Kazi Muheymin, Peter Dimopoulos, Saravanan Dayalan, Panu Phinjaroenphan, James Broberg, Sunidhi Bhalla and Vidura Abhaya, for their friendship, supports and joy during my time there. I would also like to thank Alisa Becker, Sarah Rewell, Himanshu Joshi, Brent Avery and Mikhail Perepletchikov for proof-reading my thesis.

I would like to express my love, respect and thankfulness to my father, mother, and brother. Even though they were overseas while I was writing this thesis, they always gave me the strength and confidence to overcome all the obstacles and to achieve my goals in studying and in life.

Last but not least, I would like to send my thanks to my boyfriend, Xuan Thang Nguyen for his strong support in my studies. His constant encouragement and love gave me more strength to finish this great project.

Credits

Portions of the material in this thesis have previously appeared in the following publications:

- K. A. Phan, P. Bertok, A. Fry, C. Ryan, Minimal Traffic-constrained Similarity-based SOAP Multicast. To appear in the *Proceedings of the 9th International Symposium on Distributed Objects, Middleware, and Applications*, Algarve, Portugal, November 2007.

- K. A. Phan, Z. Tari, P. Bertok, eSMP: A Multicast Protocol to Minimize SOAP Network Traffic in Low Bandwidth Environments. To appear in the *Proceedings of the 32th Annual IEEE Conference on Local Computer Networks*, Dublin, Ireland, October 2007.

- K. A. Phan, Z. Tari, P. Bertok. Optimizing Web Services Performance by Using Similarity-based Multicast Protocol. In *Proceedings of the 4th IEEE European Conference on Web Services*, pages 119–128, Zurich, Switzerland, December 2006.

- K. A. Phan, Z. Tari, P. Bertok. Enhanced SOAP Performance For Mobile Applications. In *Proceedings of the 21st Annual ACM Symposium on Applied Computing*, pages 1139–1144, Dijon, France, April 2006.

- K. Lai, K. A. Phan, Z. Tari. Efficient SOAP Binding for Mobile Web Services. In *Proceedings of the 30th Annual IEEE Conference on Local Computer Networks*, pages 218–225, Sydney, Australia, November 2005.

This work was supported by the Australian Research Council (ARC) Linkage grant No. LP0455234, entitled “Designing an Efficient and Scalable Infrastructure for Mobile Web Services” and the Distributed Systems and Networking Discipline at RMIT University.
The thesis was written in the `Eclipse` editor on Windows, and typeset using the LaTeX 2_ε document preparation system.

Contents

List of Figures

List of Tables

Abstract

Web services have been a focal point of research in the past several years. Recent advances in wireless and mobile communication and portable computing technologies have led to the emergence of mobile Web services. This area of research has gained greater importance with the increasing ubiquity of Web service applications in mobile and wireless environments. SOAP, a de-facto communication protocol of Web services, is popular for its interoperability across organisations. It is desirable that SOAP performs efficiently in environments where there are a large number of transactions. However, SOAP is based on XML and therefore inherits XML's disadvantage of having voluminous messages. When there are many transactions requesting similar server operations, using conventional SOAP unicast to send SOAP response messages can generate a very large amount of traffic [Govindaraju et al., 2004; Ng et al., 2005].

Firstly, the performance of different SOAP bindings is investigated. HTTP is the most widely used transport protocol for SOAP, however HTTP, which uses TCP, experiences high protocol overhead due to TCP's strict connection control mechanism. A benchmark of different SOAP bindings in wireless environments demonstrates the unsuitability of HTTP and TCP bindings in limited bandwidth environments. UDP is recommended as an alternative transport protocol for SOAP.

Secondly, the thesis examines the use of multicast in reducing the traffic caused by SOAP messages in low bandwidth environments to deal with challenges described. The focus is on reducing overall network traffic by optimizing the total size of messages transmitted to clients. A novel SOAP-level multicast protocol based on the similarity of SOAP messages, called SMP (Similarity-based SOAP Multicast Protocol), is proposed. In particular, issues of traffic, network optimization, response time and scalability are investigated.

Lastly, two extensions of SMP are proposed to further improve the performance of SMP. SMP's extensions are two algorithms, greedy and incremental tc-SMP, for traffic-constrained

similarity-based SOAP multicast. Tc-SMP optimizes network traffic by building its own spanning trees instead of using the one built by traditional methods, such as Dijkstra's algorithm. A new client is added to a tc-SMP tree through an existing tc-SMP node that causes minimal additional traffic for that connection.

Extensive experiments have shown that using UDP binding for SOAP results in high reduction in protocol overhead and considerable improvement in response time. Detailed analytical models and experimental evaluations of the proposed methods demonstrate that combining SOAP messages of similar content and multicasting them as aggregated messages can significantly lower total network traffic. These improvements are advantageous for Web service applications that involve a high number of simultaneous similar transactions such as stock quotes, weather and sport event reports. Such applications often generate a large amount of traffic which can put a heavy load on resource-limited environments such as mobile and wireless networks. By applying SOAP-over-UDP binding, SMP or tc-SMP to these applications, the amount of network traffic can be significantly reduced, thus freeing network capacity for other applications.

Chapter 1

Introduction

Over the past decade, we have seen phenomenal interest in the deployment of Web services in many enterprise applications. Web services are a new type of Web application based on the Simple Object Access Protocol (SOAP) that allows interoperability between different platforms, systems and applications written in different languages. The development of Web services not only attracts interest from the research community but also from industry. Most large organisations in the software, information technology and telecommunication industries have been working closely with the World Wide Web consortium to develop Web service standards such as WS-Addressing, WS-Reliability, WS-Security and WS-Management. Many organisations have achieved some degree of success, such as generating new sources of revenue or streamlining their internal and external processes when deploying Web service technologies for their enterprise services [Marshak, 2004; Microsoft Corporation Inc, 2006; 2003; Whittle, 2007].

Web service technologies, such as SOAP, Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI), promise to provide seamless integration of services provided by different vendors in different industries and written on various platforms and languages. Examples of such services include travel booking, real-time stock quotes, currency exchange rates, credit card verification, driving directions and yellow pages. A popular example used to illustrate Web services is the travel reservation application. A travel agent company offers a complete vacation package including airline/train/bus tickets, hotel reservation, car rental and tours. This service involves many service providers such as airlines, transport companies, hotels, tour organizers and credit card companies for payment. The back-end of service applications offered by these service providers are likely to be written in different programming languages (e.g. C++, Java, .Net) and implemented on different platforms (e.g. Linux, Macintosh and Windows). But if all of these services are implemented using Web service technologies and are published on a public registry such as the UDDI [OASIS, 2006a], the travel agency can search all services from the one platform. Consumers who want to book vacation packages can come to the travel agency's website and specify some criteria such as location, means of transport and price range for their travel. The travel agency will act on behalf of the consumer and search for appropriate services registered on the UDDI and return results that satisfy the user's requests.

Obviously the application explained above can be implemented using traditional distributed computing technologies such as Distributed Component Object Model (DCOM) [Horstmann and Kirtland, 1997] and Common Object Request Broker Architecture (CORBA) [OMG, 2001]. However, such conventional techniques do not provide the same high level of interoperability that Web services do. In particular, if an application were to be developed using DCOM, all participating nodes in the distributed application would have to be running on Windows platform [Gisolfi, 2001]. CORBA is based on the object-oriented model and a binary transport, Internet Inter-ORB Protocol (IIOP) [OMG, 2007], hence an Object Request Broker (ORB) node assumes a certain representation exists in other nodes to allow them understand each other. SOAP endpoints are, on the other hand, not dependent on any specific data representation or platform, since all data is already formatted in a high level language, namely XML.

With the exciting prospects of what Web service technologies can bring come many difficult challenges. Web services' major problem is that they generate a large amount of network traffic. This comes from the fact that Web services are based on SOAP, an XML-based communication protocol. SOAP provides the basic messaging infrastructure for Web services by exchanging XML messages. It is XML's textual representation and redundant characteristics that cause the major performance bottleneck. Tian et al. [2004] performed a test showing the number of additional bytes Web services generate. There were 589 bytes in both request and response messages for a service requesting the details of a book given an ISBN in the parameter of the request. But more than 3900 bytes had to be sent when using SOAP, while only 1200 bytes were sent when traditional Web interaction with HTML was used.

SOAP's overhead stems mainly from the use of XML. Since both SOAP and WSDL are XML-based, XML messages have to be parsed on both the client and the server side. XML parsing occurs at run time, therefore the required additional processing time results in longer total response time. The limitations of SOAP for scientific computing were investigated by Davis and Parashar [2002]. Their experiments compared SOAP with Java RMI by sending large arrays of doubles; the results showed that SOAP was slower than Java RMI by a factor of ten. in benchmarking chapter An experimental evaluation of SOAP performance in business applications was presented by Kohlhoff and Steele [2003]. In this work, SOAP was compared with Financial Information eXchange (FIX) protocol which also used a text based wire representation as SOAP, and with Common Data Representation (CDR), a common binary wire format. The results demonstrated that the text-based protocols (SOAP and FIX) have slightly lower performance than the binary protocol (CDR) due to the complexity of the XML syntax.

Despite the rapid growth in wired network bandwidth and steady increase in wireless bandwidth with new mobile technologies such as 3G networks, it is still not infinite. The available network bandwidth is often limited and expensive, especially in mobile and wireless environments. Enterprise IT systems need to process thousands of Web service requests in a short period of time. Considerable increased traffic represents high consumption of the network resources. Web services are promised to be a source of generating increased revenues for enteprises by exposing existing enterprise applications to a wide range of other applications on different platforms. High network traffic can hold up this potential for revenue generation and needs to be addressed. It is important to design Web services that have low communication overhead and make efficient use of available bandwidth.

1.1 Summary of Existing Solutions

Several solutions have been proposed to improve SOAP performance, either using binary encoding (binary XML instead of textual XML), caching (at the client side by increasing the locality of objects), compression (by reducing the size of XML payload) or optimizing the SOAP run-time implementation (by efficient optimization of the kernel). In this section, important related studies on Web service performance enhancements are briefly reviewed.

One type of solution attempts to reduce the size of SOAP messages by binary encoding (similar to CORBA encoding). It is transmissions of SOAP messages in binary instead of textual format. Generic SOAP engines support both textual and binary XML as the encoding scheme of messages. Scientific data could be directly transmitted as binary XML. Lu et al. [2006] developed a binary XML encoding scheme called BXSA (Binary XML for Scientific Applications). BXSA supports the ability to convert a textual XML document to binary

XML and vice versa. A SOAP message is modeled in the bXDM model (a scientific-data-friendly XML data model that is proposed by the same authors and is extended from the XPath Data Model [Fernandez et al., 2007]) instead of the XML Infoset. To send a SOAP message, first a SOAP message is constructed in the bXDM model, then the encoding policy provider is invoked to serialise the message into an octet stream. Finally, the stream is transferred by calling the binding policy provider. The reverse procedure takes place when a message is received. Both SOAP over BXSA/TCP scheme and SOAP with HTTP data channel have similar performance. They can rebind the BXSA transport to multiple TCP streams, thus it can carry larger messages.

W3C XML Protocol Working Group recently released specifications for SOAP Message Transmission Optimization Mechanism (MTOM) [Gudgin et al., 2005a] and XML-binary Optimized Packaging (XOP) [Gudgin et al., 2005b]. These specifications are targeted to multimedia data (such as JPEG, GIF and MP3) and data that includes digital signatures. The specifications define an efficient means of XML Infoset serialization. An XOP package is created by placing a serialization of the XML Infoset inside an extensible packaging format such as MIME [Gudgin et al., 2005b]. MTOM describes how XOP is layered into the SOAP HTTP transport. However, XOP and MTOM still possess a parsing issue inherited from SOAP and XML.

Another example of work in binary SOAP encoding is a study by Oh and Fox [2005]. They proposed a new mobile Web service architecture, called Handheld Flexible Representation (HHFR), that provides optimised SOAP communication using a binary message stream. HHFR architecture separates XML syntax of SOAP messages from SOAP message contents. This separation is negotiated at the beginning of a stream. An XML schema is used to characterize the syntax of the SOAP body. HHFR is most suited to Web service applications where two end-points exchange a stream of messages, because messages in a stream often share common structure and type information of the SOAP body and most parts of the SOAP headers. The message structure and type in form of XML schema are transmitted only once and the rest of the messages in the stream have only payloads. Oh and Fox compared HHFR prototype with a conventional SOAP and found the higher performance advantage of HHFR is achieved when there are multiple messages transmitted in a session. In particular, HHFR streaming communication outperforms conventional SOAP by 7 times in round trip time for a service adding float numbers.

Compression is a popular method to deal with large message sizes of Web services. Compression is particularly useful for poorly connected clients with resource-constrained devices or for clients that are charged by volume and not by connection time by their providers. However, compression decreases server performance due to the additional computation required. From experiments of XML compression in wireless networks, Tian et al. [2004] found that in a low bandwidth network such as GPRS the service time was halved when compressing large SOAP responses. The response time during overload is however about 40% higher and the server throughput is about 50% lower when compression is used. Therefore, Tian et al. proposed that clients should decide whether they want their responses compressed. During low server demand, responses to all client requests except those that did not ask for compression are compressed. During high server demand, only responses to clients that asked for compressed responses are compressed. Despite high response time and low throughput, Tian et al. have shown that their dynamic compression approach is beneficial for both the server and for mobile clients with poor connectivity. It is also recommended that servers should only compress replies to clients that can benefit from compression.

Many studies have researched approaches to enhance SOAP performance through caching [Devaram and Andresen, 2002; Liu and Deters, 2007; Takase and Tatsubori, 2004; Terry and Ramasubramanian, 2003]. Devaram and Andresen [2002] implemented a partial caching strategy to cache SOAP payloads on the client side. In this method, the SOAP payload is cached when it is first generated. Every time the client makes a request, the payloads stored in the cache are reused to create a new payload by replacing some values of the XML tags with new parameter values. This technique is shown to provide better performance than non-caching for request messages with small number of tags. The performance of the partial caching technique degrades when there are many parameters defined in a SOAP request because the time spent on substituting the parameter values and accessing file I/O increases as the number of parameters increases, which in turn enlarges the size of the cache.

The advantage of Web service caching is mainly in supporting disconnected operations. Terry and Ramasubramanian [2003] implemented a HTTP proxy server between a Web service provider and a Web service consumer to provide a simple cache for storing SOAP messages. Their study highlighted the benefits of employing a Web service cache to support disconnected operations. Specifically, in case of disconnection, SOAP response messages that are stored in the cache, will be returned to client requests. The SOAP requests are stored in a write back queue which is later played back to the server when the connection to the Web service is restored. However, there are still many issues with caching such as consistency and availability of offline access to Web services. Another difficulty with Web service caching is that a cache manager does not know which operation needs to be played back to the server.

In addition, the effectiveness of a cache is often dependent on the similarity of future requests to past requests.

Liu and Deters [2007] proposed a dual caching strategy for mobile Web services. In their method, one cache resides on the client side and the other on the server side to handle any problems due to loss of connectivity during the sending/receiving of request and response messages. The two caches are coordinated by a cache manager. An ontology Web language is used to describe meta-data used on the caches such as service description, client workflow description and connectivity description. This ensures interoperability with other Web service standards. In fact, Terry and Ramasubramanian [2003] also emphasize the importance of understanding the cacheability of services in their work. Therefore, they propose to add annotations in the WSDL specification to support SOAP caching. The suggested annotations include semantic information such as cacheability, life time, play-back and default-response. This however leads to issues regarding standards and interoperability.

In cases when outgoing SOAP messages are very similar in content, it is advantageous to use differential encoding. With this technique, only the difference between a message and a previous one is sent over the wire. Documents containing only the differences can be more compact in size depending on content. The important studies in the differential encoding area are a research of Abu-Ghazaleh et al. [2004] on a differential serialization technique on the server side , a study of Suzumura et al. [2005] on differential deserialization and a work of Werner et al. [2005] on differential compression.

1.2 Important Research Questions

As explained above, there have been several key studies into the improvement of SOAP performance. Each of the above techniques has its own advantages in improving SOAP performance; however, they focus on the SOAP engine, compression and caching and fail to look at other aspects (like SOAP binding and networking). Little work has been done on the use of alternative binding options for SOAP to handle the inherent performance issues of wireless environments and on multicasting SOAP messages to reduce network traffic.

The SOAP binding specifies which underlying protocol to be used to deliver a SOAP message [Ferris and Williams, 2001]. HTTP is the most widely used protocol for SOAP binding, however, there is high overhead associated with HTTP in wireless networks. Recently, the SOAP-over-UDP specification [BEA Systems Inc. et al., 2004] has been proposed to provide basic guidelines on how to transmit SOAP messages over UDP; however, it does not cover the binding in wireless environments and for resource-constrained devices.

Multicasting is a well known technology for conserving network bandwidth. Multicast is often employed in applications where the same data is transmitted to a group of interested clients. Instead of sending replicated unicast packets of the same data to multiple clients, multicast reduces the number of packets sent over links, thus reduces the use of network resources. Existing work in IP, application and content-based multicasts [Oliveira et al., 2005; Pendarakis et al., 2001; Shah et al., 2004; Zhang et al., 2002] have shown that the use of multicast utilizes network bandwidth efficiently and can reduce delivery delay.

SOAP can benefit from the use of multicast as well because duplicate large SOAP messages can be avoided. This in turn reduces SOAP serialisation time on the server side and consumes less network resources. Previous studies on multicast were based on the assumption that multicast messages are identical. The work presented in this thesis is however not based on this assumption, but examines how similar (not necessarily identical) SOAP messages can be multicast together in an aggregated message represented in a special schema XML format. This will result in a further reduction in the total size of messages sent over a network because only one copy of matching parts within SOAP messages is sent. In particular, the feasibility of adapting a similarity-based multicast protocol for SOAP to reduce network traffic in low bandwidth environments is studied. This thesis also looks at the analytical and experimental analysis of some existing solutions and compare them to the proposed solutions. The similarity measurement of SOAP messages plays a key role in the proposed SOAP multicast protocol. In particular, the following main research questions are pursued:

1. What are the performance limitations of current SOAP HTTP binding, particularly in supporting mobile Web services? What is an effective SOAP binding option for wireless environments? This thesis will deal with a performance benchmark which compares three implementations of SOAP binding options: SOAP-over-HTTP, SOAP-over-TCP and SOAP-over-UDP.

2. What is a cost-effective method to reduce total traffic created by SOAP messages sent over a network? Techniques for reducing SOAP message size have widely been studied in the context of Web services [Abu-Ghazaleh and Lewis, 2005; Takase and Tatsubori, 2004; Tian et al., 2004; Werner et al., 2005]. However the results from these approaches are limited to reducing traffic at client or server side during caching or serialisation; and are dependent on the similarity of future requests to previous ones. Can multicasting SOAP messages lead to improved performance? Can multicast take

advantage of SOAP's nature of having messages with similar data structure? What is an effective model to measure the similarity between SOAP messages?

3. What are the most efficient routing algorithms that can be used to deliver similar SOAP messages so that the total traffic created over a network can be minimised? While routing algorithms (such as Dijkstra's and Bellman-Ford's algorithms) are widely used on the Internet, they consider only a simple cost metric (such as hop counts). Other Quality-of-Service (QoS) metrics (such as network bandwidth, network traffic and end-to-end delay) are not taken into consideration. Although there is a wide range of studies in the QoS routing area [Chen and Nahrstedt, 1999; Guo and Matta, 1999; Yuan, 2002; Zhu et al., 1995], little work has been done in developing appropriate QoS routing algorithms for SOAP traffic. What are approaches for QoS-based SOAP multicast routing? What are the trade-offs with the proposed algorithms?

1.3 Research Contributions

In addressing the above research questions, this thesis makes a number of contributions to advance the current state of the art of research in SOAP communication performance. These contributions are summarized below.

A) SOAP Binding Benchmark

A benchmark of different SOAP bindings in wireless environments is proposed and implemented. Its configuration and results can serve as a standard benchmark for other researchers who are also interested in the performance of SOAP bindings in wireless networks. Three sets of experiments were carried out: loopback mode, wireless network mode and mobile device mode. The experimental results show that HTTP binding inherits very high protocol overhead (30%–50% higher than UDP binding) from TCP due to the slow connection establishments and tear-down processes and the packet acknowledgement mechanism. UDP binding has the lower overhead because it does not require establishing connections before transmitting datagrams and does not address reliability. This results in a reduction in response time and an increase in total throughput.

B) Similarity-based SOAP Multicast Protocol

A novel SOAP multicast technique [Phan et al., 2006b], called SMP (Similarity-based Multicast Protocol) is proposed, which takes into account the similarity of SOAP messages. SMP exploits the feature of similar data structure among SOAP messages to group messages of similar content in one aggregated message (called SMP message) so that common data types or values are not repeated and are sent only once. A similarity measurement model for SOAP messages is proposed. The server must establish the similarity of outgoing messages in order to decide which messages can be aggregated to improve the overall performance without incurring high communication costs. A SOAP message indexing technique is proposed to represent SOAP messages in a special XML format, so a more compact representation can be used to reduce more traffic. This indexing technique is based on the data type definitions contained in WSDL service description. Each XML node in an indexed version of a SOAP message is composed of the node's data type ID, which is referenced back to the WSDL document, the position of the node in breadth first search traversal, and the node value. The SOAP message index assists in fast merging of SOAP messages and splitting of SMP messages because it enables easy grouping of common and distinctive data in SMP messages.

C) SOAP Multicast Protocol to Minimize SOAP Network Traffic

The original proposal of SMP uses the Open Shortest Path First (OSPF) routing protocol [Cormen et al., 2001] to send SMP messages. Under SMP, the more similar SOAP messages can be aggregated over common links the more network bandwidth can be saved. However, when the OSPF protocol is used, some SOAP messages that are very similar in content will follow paths that may not share many common links. To deal with such a problem, an extension of SMP, which is the traffic-constrained similarity-based SOAP multicast protocol (tc-SMP) is proposed here. Two algorithms, greedy and incremental approaches, are proposed to address this problem [Phan et al., 2007a;b]. Both tc-SMP algorithms aim at minimizing the total network traffic of the whole routing tree every time a new client is added to the tree. Two heuristic methods are also proposed for these algorithms to assist in choosing the order of clients being added to the tree. In general, the performance improvement of tc-SMP is about 30% higher network traffic reduction than SMP at a small expense of up to 10% rise in response time.

1.4 Thesis Organization

The rest of the thesis is organized as follows.

- Chapter 2 provides an overview of areas needed to understand the core chapters of this thesis. Important Web service concepts and standards, SOAP technologies, traditional multicast protocols and popular routing protocols on the Internet are discussed.

- Chapter 3 details a benchmark suite of various SOAP binding options. Three extensive sets of experiments are described to highlight the advantages of using SOAP-over-UDP binding in mobile and wireless networks.

- Chapter 4 addresses the problem of high SOAP traffic in limited bandwidth environments by proposing a similarity-based SOAP multicast protocol (SMP). A model for measuring the similarity of SOAP messages is proposed. SMP routing algorithm is described in detail with its analytical model and experimental evaluation.

- An extension of SMP protocol, traffic-constrained SMP (tc-SMP), is described in Chapter 5. Two new source routing algorithms, greedy tc-SMP and incremental tc-SMP, are proposed to send SMP messages along paths that highly similar messages have more common routing links. The tc-SMP routing algorithms are used instead of the OSPF protocol to convey SMP messages. Both theoretical and experimental studies of tc-SMP are discussed.

- Finally, the thesis concludes in Chapter 6, where the main contributions are summarized and possible directions for future research are discussed.

Chapter 2

Background

In this chapter, background material to facilitate the understanding of the work in this thesis is presented. There are several XML-based standards in the area of Web services. A brief survey of the most important standards needed to understand this work will be presented. In addition, traditional multicast routing protocols and network routing algorithms will be reviewed to provide background knowledge for the next two chapters.

2.1 Web services

Web services have emerged as a key technology that enables interoperability between distributed applications. A Web service is a reusable piece of software that interacts with clients, possibly with other Web services by exchanging messages over the network which comply Extensible Markup Language (XML), SOAP and other industry recognized standards. From a different perspective, a Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion WSDL which is called its service description. A WSDL description covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented, and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented and be cross-technology implementations.

2.1.1 Web Service Architecture

The Web services architecture is defined using the Service Oriented Architecture (SOA) pattern. SOA is a component model that inter-relates different functional units of an application, called services, through well-defined interfaces and contracts between these services. The interface is defined in a neutral manner that should be independent of the hardware platform, the operating system, and the programming language the service is implemented in. SOA defines three roles: a service requestor, a service provider and a service registry. The three main roles and the interaction between the roles are depicted in Figure 2.1.

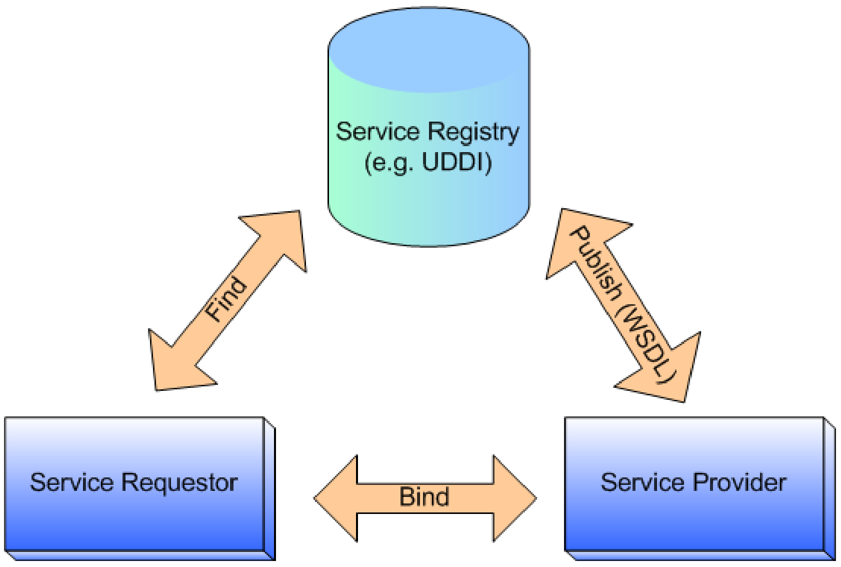


Figure 2.1: Web service architecture

- A *Service Provider* is responsible for creating a service description, deploying that service in a runtime environment that makes it accessible by other entities over the network, publishing that service description to one or more service registries, and receiving Web service invocation messages from one or more service requestors.
- A *service requestor* (or service consumer) is responsible for finding a service description published to one or more service registries and is responsible for using service descriptions to bind to or invoke Web services hosted by service providers.
- A *service registry* (or service broker) is responsible for advertising Web service descriptions submitted to it by service providers. It also allows service requestors to search the

collection of service descriptions contained within the service registry by replying to the queries from the service requestor on the availability of the service and the quality of service (QoS) provided by an available service. Universal Description, Discovery and Integration (UDDI) [OASIS, 2006a] is an example of the most popular service registry for Web services currently.

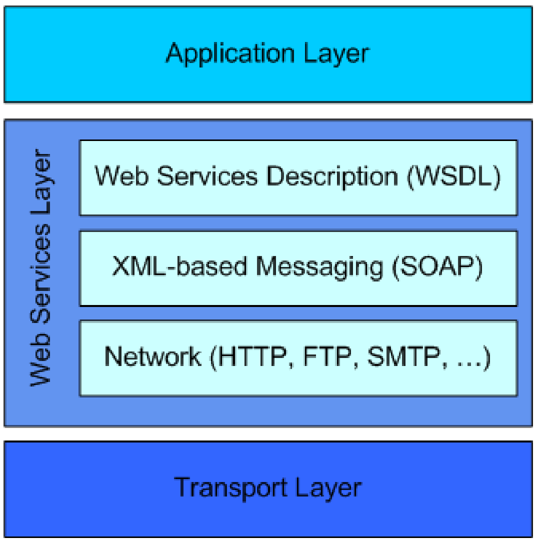

Application Layer
Web Services Layer
Web Services Description (WSDL)
XML-based Messaging (SOAP)
Network (HTTP, FTP, SMTP, ...)
Transport Layer

Figure 2.2: Web service protocol stack

The Web service layer is placed between the transport layer and the application layer in the Internet reference model as described in Figure 2.2. Within the Web service layer, the network protocols such as HTTP (Hypertext Transfer Protocol), SMPT (Simple Mail Transfer Protocol), FTP (File Transfer Protocol) and BEEP (Blocks Extensible Exchange Protocol) are at the bottom. HTTP is the de-facto transport protocol for Web services because if its ubiquity and ability to pass through firewalls. However, any other transport protocols, such as TCP, UDP, SMTP and FTP could be used instead. The XML-based SOAP forms the next layer. WSDL is in the top layer.

2.1.2 Web Service Description Language (WSDL)

Service description is a key feature within a service-oriented architecture (SOA). A service description is involved in each of the three operations of SOA: *publish*, *find* and *bind*. Referring back to Figure 2.1, the service provider publishes the service description to the service registry during the *publish* process. During the *find* operation, the service requestor searches available service descriptions in the service registry to identify a matching service. The service description also defines the message format expected by the service provider so that service requestor can send request messages that can be understood by the provider during the *bind* operation.

The service description of a Web service is defined by using Web Service Description Language (WSDL) [Christensen et al., 2001]. WSDL is an XML-based language for describing the technical specifications of a Web service. It describes the operations offered by a Web service, the syntax of the input and output documents, the communication protocol to use for communication with the service and the location of the service. Appendix A provides the WSDL description of a Stock Quote service.

2.2 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) [Gudgin et al., 2007] is a standard for Web services messaging. SOAP was designed to replace traditional remote communication methods such as DCOM, CORBA and RMI. The main benefit of SOAP is interoperability. It allows applications written in different languages and deployed on different platforms to communicate with each other over the network. SOAP uses XML technologies to define an extensible messaging framework, that provides a message construct that can be exchanged over a variety of underlying protocols (such as HTTP/HTTPS, TCP, UDP, BEEP and SMTP). Thus, SOAP creators have defined a binding framework for SOAP instead of a fixed binding. Specifically, the SOAP binding framework specification [Ferris and Williams, 2001] provides a high level of flexibility in terms of how SOAP messages are transmitted.

SOAP is fundamentally a stateless, one-way message exchange paradigm, which can be used as a building block for creating more complex interaction patterns such as one-way, request/response, notification and notification/responses, by combining one-way exchanges with features provided by an underlying protocol or application-specific information. SOAP does not dictate the semantics of any application-specific data that it conveys, such as the routing of SOAP messages, reliable data transfer and firewall traversal. However, SOAP provides the framework by which application-specific information may be conveyed in an extensible manner. Also, SOAP provides a full description of the required actions to be taken by a SOAP processor node on receiving a SOAP message.

```
POST /axis/servlet/AxisServlet HTTP/1.0
Content-Type" text/xml; charset=utf-8
Accept: application/soap+xml
User-Agent: Axis/1.2RC3
Host: http://www.weatherhost.com:8081
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 438
Authorization: Basic dXN1cjE6cGFzczE=

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope: xmlns:soapenv="..."
                   xmlns:xsd="..."
                   xmlns:xsi="...">
   <soapenv:Body>
     <ns1:getTemperature soapenv:encodingStyle="..."
                         xmlns:ns1="...">
        <symbol xsi:type="xsd:string">
           Melbourne
        </symbol>
     </ns1:getTemperature>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 2.3: Sample SOAP request message

2.2.1 SOAP Message Structure

Figure 2.5 shows the basic structure of a SOAP message consisting of three parts: an envelope, an optional header, and a mandatory body. The root element of a SOAP message is an Envelope element containing an optional *header* element for SOAP extensions and a *body* element for payload. The header element of a SOAP message may include implementations of SOAP extensions such as Web Service Addressing [W3C, 2004], Web Service Security [OASIS, 2006b], Web Service Reliable Messaging [Bilorusets et al., 2005]. The body construct of a SOAP message acts as a container for the data being delivered by the SOAP message. SOAP offers a standard encoding style (serialization mechanism) to convert arbitrary graphs of objects to an XML-based representation, but user-defined serialization schemes can be used as well. Figures 2.3 and 2.4 provide an example of a full HTTP SOAP request and response messages of a getTemperature service using Axis SOAP implementation [Apache Software Foundation, 2007a].

```
 HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Sat, 1 Jan 2005 00:00:00 GMT
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope: xmlns:soapenv="..."
                   xmlns:xsd="..."
                   xmlns:xsi="...">
   <soapenv:Body>
     <ns1:getTemperatureResponse
          soapenv:encodingStyle="..."
          xmlns:ns1="...">
          <getTemperatureReturn href="#id0"/>
     </ns1:getTemperatureResponse>

     <multiRef id="id0" soapenc:root="0"
         soapenv:encodingStyle="..."
         xsi:type="xsd:float"
         xmlns:soapenc="...">
         23.5
     </multiRef>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 2.4: Sample SOAP response message

2.2.2 SOAP Extensions

SOAP extensions allow developers to augment the functionality of a Web service by altering the SOAP message sent to and from a Web service provider or consumer. For example, authentication, encryption or compression algorithms can be implemented to run with an existing Web service. The SOAP extension can be done during the AfterSerialize and BeforeDeserialize stages [Gudgin et al., 2007]. For example, encrypting can be done in the AfterSerialize and decrypting can be done in the BeforeDeserialize stage. It is important to note that a SOAP extension that performs modification on a SOAP message must be done both on the client and the server.

2.2.3 SOAP Message Exchange Model

SOAP messages are primarily one-way transmissions. However, multiple messages can be combined to form message exchange patterns such as request/response pattern. A SOAP processing model includes an originator, one or more ultimate destinations, and none or more

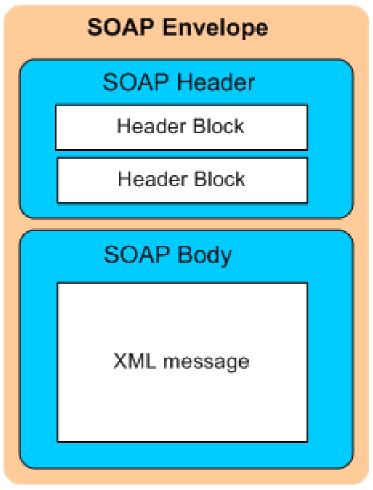


Figure 2.5: SOAP envelope

intermediaries. This model supports distributed message processing which is an advantage over the client-server messaging model.

Typically when a SOAP node receives a SOAP message, the following actions are performed:

- Identify all mandatory header blocks intended for the node
- If there is any mandatory block identified in the preceding step that is not understood by the node, stop processing; otherwise process all the header blocks that are supported.
- If the current SOAP node is not the ultimate recipient of the message, remove all SOAP header blocks identified in the first step before forwarding it along the message path. At this stage, some new SOAP header blocks may be inserted into the SOAP message. If the node is the final destination, process the SOAP body.

A message exchange pattern (MEP) describes the sequence of messages exchanged between a service provider and a service consumer. SOAP supports two basic types of message pattern: single-message exchange and multiple-message exchange. The classification of each pattern is dependent on whether the provider or the consumer is the first party to initiate the message exchange and whether one side expects a response message to the initial message. There are two basic SOAP message exchange patterns defined in the SOAP Version 1.2 specification [W3C, 2007a]:

- Request-Response MEP: is a pattern for the exchange of two messages between two adjacent SOAP nodes along a SOAP message path. Typically, a request message is first transferred from a requesting SOAP node to a responding SOAP node. Upon successfully processing the request, the responding SOAP node sends a response message back to the requesting node.

- Response MEP: is a pattern for the exchange of a non-SOAP message acting as a request followed by a SOAP message acting as a response. A request that does not contain a SOAP envelope is transmitted to a receiving SOAP node. A response message which includes a SOAP envelope is then sent to respond back to the requesting node where the processing of the SOAP envelope occurs.

2.2.4 SOAP Messaging Styles

There are two SOAP messaging styles: Remote Procedure Call (RPC) style and document style. The RPC style is usually synchronous that is a client sends a message to a server and waits to get a response or a fault message back from the server. Under an RPC-style Web service implementation, a function on a remote machine is invoked as if it were a local function. The sender and receiver communicate with each other via an interface understood by both parties [Englander, 2002]. Such an interface consists of a method name and a parameter list. The parameter list is composed of the variables passed to the called procedure and those returned as part of the response. All of the serialization and deserialization of data is handled by SOAP standards. For example, part 2 of the SOAP version 1.2 specification defines the rules to encode RPC method calls and responses as XML elements [W3C, 2007a]. With document style messaging, it is up to developers to decide how the data is represented in XML. This gives developers flexibility in choosing the schema for validating the document's structure and the encoding scheme for interpreting data item values. Under a document-style Web service implementation, a client uses an XML parser to create an XML document and then inserts it into a SOAP message's body. The client serializes the message and sent to the server. A reverse process takes place on the server side.

RPC-style messaging's main strength is that it maps closely to an object-oriented model, hence it is a good option for creating new components and for creating interfaces between existing components and Web services. Secondly, it offers a standard-based and platform-independent component technology which allows clients and servers use different programming languages to implement their respective side of the interface. However, the messaging process in RPC-style is tightly coupled on the programmable interface. Changes on this interface would require changes on both sides of the interface. In contrast, with document-style messaging the rules are less strict and enhancements can be made to the XML schema without breaking the calling application [McCarthy, 2002]. This advantage comes from the fact that in document-style an XML document is sent rather than a structured return value. Because of this nature, document-style messaging is also ideal for passing complex documents such as customer orders and invoices. Document-style messaging's drawback is that there is no standard service identification mechanism in place. The client and server must agree on a common way for determining which service needs to process a received document.

Both messaging styles suffer the same overhead in serialisation. Parsing XML documents is required on both client and server sides. In addition to the cost of XML parsing, there is the cost of carrying encoded data values, which can be much larger in size than its binary equivalent, across the network.

2.2.5 XML Similarity Measurements

Similarity is an important concept used to determine the syntactic relationship between two ore more SOAP messages. In this section, existing tools and models for similarity measures are presented. Similarity measures for ontological structures, web data or XML documents have been widely researched in the software engineering, document management and database communities [Dorneles and et. al., 2004; Ganesan et al., 2003b].

APPROXML Tool

APPROXML [Damiani et al., 2002] is a software tool for making XML pattern-based search queries to locate XML data items that are similar to a searched pattern. In this tool, XML documents are represented as graphs using the DOM model. Each edge of a document is weighted to express their importance. The weighting technique takes into account the various characteristics of each edge. Multiple weights on each edge are then aggregated in a single arc-weight.

A searched XML pattern is a partial subtree. APPROXML scans the graph data set searching for subgraphs matching the pattern supplied by the user. The tool uses the edge weights to compute the match value for each hit, and returns a list of results sorted according to the similarity level between the found subgraph and the searched pattern.

Subtree Matching in Approximate XML Joins

Another important work in XML matching is from Liang and Yokota [2005]. Liang and Yokota proposed approximate XML join algorithms based on leaf-clustering for measuring the similarity between XML documents. The two XML documents to be joined are segmented into subtrees. The similarity degree between the two subtrees is determined by the percentage of the number of matched leaf nodes out of the number of leaf nodes in the base subtree.

However, with this solution the one-to-multiple matching problem may occur when there are more than one subtrees which have the same similarity degree with the base subtree. Liang and Yokota then extended their work to propose a path-sequence based discrimination method Liang and Yokota [2006] to determine the most similar one from several matched target subtrees. According to their definition, a path sequence of a pair of matched subtrees is the path from the root node to the matched leaf in either the base or target subtree. For a pair of matched leaves, the path-sequence similarity degree is the percentage of the number of nodes in the base path sequence that have the same labels or values with those in the target path sequence; and the total number of nodes in the base path sequence.

Jaccard's Coefficient

Different measurement methods have different ways to normalize the intersection values. One of the most popular measures is the *Jaccard's coefficient* [Jaccard, 1901].

Definition 1 *Jaccard's Coefficient*: Given two sample sets X and Y, their similarity is defined as:

$$sim_{Jacc}(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}, \text{ where}$$

- $X \cap Y$ is the intersection of sets X and Y;
- $X \cup Y$ is the union of sets X and Y; and
- $|A|$ is the cardinality of set A.

Vector Space Model

Intersection-based measures do not accurately capture the similarities in certain domains, such as when the data is sparse or when there are known relationships between items within sets. The *Vector-Space* model is another popular model, especially in the information retrieval domain, where each element is modeled as a dimension in a vector space [Ganesan et al., 2003a]. A collection of elements is then represented by a vector, with components along the dimensions corresponding to the elements in the collection. The advantage of this technique is that weights can be assigned to the components of the vector.

2.3 Multicast Protocols

The proposed SOAP multicast routing protocol, presented in this thesis to improve Web services performance, is based on the similar structure of different SOAP message instances. It is beneficial to give, in this section, an overview of different existing multicast solutions and to evaluate if they are suitable for multicasting Web services. Firstly, the characteristics of multicast applications are explained, followed by a discussion on the strengths and weaknesses of four various types of multicast protocols — IP multicast, application layer multicast, content-based multicast and explicit multicast protocols.

Theoretically, any application in which more than one participant shares some common data can be designed using multicast. Multicasting is suitable for the following types of applications: group type activities, file transfers, electronic distribution of software, video conferences, white-boards and live broadcasts [Radha et al., 2004]. This section focuses on distribution-based type of multicast applications that are suitable for Web services deployment. Unlike broadcasting, where a message is sent to all clients, or replicated unicasting, in which messages are sent one by one to each client, multicasting involves sending messages to only a group of interested clients. Hence multicasting can be expected to reduce use of network resources.

2.3.1 IP Multicast Protocols

Traditionally, IP multicast routing protocols are used to perform multicast at the IP layer. IP multicast consists of a group of participants called multicast group, of which there is typically one source, many receivers, and a set of intermediate routers. The source sends the same information to all receivers by setting up a multicast tree. Each intermediate router in a multicast tree needs to support multicast in order to recognize a multicast packet, process and route it to its children nodes. Receivers use a group membership protocol to inform the network when they wish to join a group [Fenner, 1997]. The network, in turn, runs a multicast routing protocol that is distinct from the unicast routing protocol. The former is used to build and maintain a distribution tree rooted at the source with branches that take the shortest, loop-free paths down to sub-networks where group members exist. A router on a multicast tree with two or more separate downstream links is responsible for copying the packet and transmitting it down each link. Figure 2.6 illustrates a typical multicast tree.

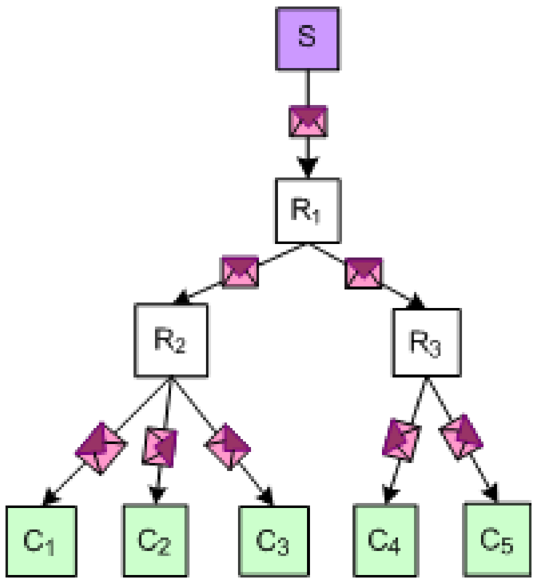


Figure 2.6: A Simple IP Multicast Tree

Receivers C_1 to C_5 use a group membership protocol to inform the network when they wish to join a multicast group. This protocol is also used to build and maintain a distribution tree rooted at the source S. An intermediate router with two or more separate downstream links such as R_1 is responsible for copying the packet sent by the source and transmitting it down to R_2 and R_3. This process continues until all members of the multicast group receive the packet. With IP multicast, there are a total of 8 packets sent on 8 links; if unicast is used instead, 15 packets would be sent (the source needing to send a copy of the message to each client individually.) Thus, IP multicast significantly reduces the network traffic.

The most popular IP multicast protocols are Distance Vector Multicast Routing Protocol (DVMRP), Multicast Extension to OSPF (MOSPF), and Protocol Independent Multicast (PIM). DVMRP and MOSPF perform well if group members are densely packed. However, DVMRP periodically floods the network and MOSPF sends group membership information over the links [Boudani et al., 2004], so these methods are not efficient in cases where group members are sparsely distributed among regions and the bandwidth is limited. Due to this scalability problem, most internet service providers rely on PIM-Sparse Mode (PIM-SM), which is adapted to groups where members are sparsely distributed [Boivie et al., 2000].

One problem with IP multicast protocols is that routers have to keep a forwarding state for every multicast tree that passes through it [Boudani et al., 2004]. Thus these protocols suffer from scalability problems for high numbers of concurrently active multicast groups. However, the use of state maintenance contradicts the stateless philosophy of IP routing which requires network routers keep minimum possible state information for routing purposes.

2.3.2 Application Level Multicast Protocols

There are some applications, such as video conferences, multi-player games, private chat rooms and web cache replication, whose requirements are substantially different from the design point of IP multicast. Those applications contain small groups with few members and the groups are often terminated dynamically; the number of groups that are concurrently active may be large. For a large number of such small and sparse groups, the benefits in terms of bandwidth efficiency and scalability of IP multicast are often outweighed by the control cost and complexity associated with group set-up and maintenance [Radha et al., 2004]. In these cases, there is a need for multi-sender multicast communication which scales well for a large number of communication groups with small number of members and does not depend on multicast support at routers.

Application layer multicast, a well-studied problem in the context of content distribution networks, provides multicast functionality at the application layer while assuming only unicast IP service at the network level. A range of research has addressed this area. Notably, an application level group multicast protocol, called ALMI [Pendarakis et al., 2001], allows a simplified network configuration without need of network infrastructure support. ALMI takes a centralized approach to tree creation. Members of a multicast group perform network measurements between themselves as a measure of distance. A controller collects these measurements from all members, computes a minimum spanning tree based on measurements and then disseminates routing tables to all members.

In contrast to ALMI's centralized approach, Zhang et al. in [Zhang et al., 2002] propose a Host Multicast Tree Protocol (HMTP), which is a tree-based end-host multicast protocol. HMTP builds a group-shared tree instead of a source-specific tree. The deployment of the IP multicast protocol has been limited to islands of network domains under single administrative control. HMTP automates the interconnection of IP-multicast enabled islands and provides multicast delivery to end hosts where IP multicast is not available. With HMTP, end-hosts and proxy gateways of IP multicast-enabled islands can dynamically create shared multicast trees across different islands.

Each group member runs a daemon process (Host Multicast agent) in user space. The daemon program provides Host Multicast functionality at the end-host. An IP multicast island is a network of any size that supports IP multicast. Within an island, native IP multicast is used to send and receive data. One member host in an island is elected as the Designated Member (DM) for the island. Different islands are connected by UDP tunnels between DMs. Data encapsulated in UDP packets flow from one island to another through the tunnels. Upon reaching an island, the data packets are de-capsulated by the DM, and then multicast onto the island. Zhang et al.'s simulation results show that the host multicast tree has low cost and that data delivered over it experiences moderately low latency. HMTP supports the IP multicast service model and it automatically uses IP multicast where available. Thus, it takes advantages of the scalability of IP multicast, making HMTP itself more scalable.

2.3.3 Content-Based Multicast

A disadvantage of IP multicast is that IP multicast services do not consider the structure and semantics of the information being delivered. Especially for multicasting personalized information such as delivering country music but except some songs or artists to a recipient, traditional IP multicast does not utilize network bandwidth efficiently because full information is delivered to the recipient and filtering is done at the recipient end. Shah, Ramzan and Dendukuri in [Shah et al., 2004] proposed the use of content-based multicast (CBM) in which extra content filtering is performed at the interior nodes of the IP multicast tree. If the filtering process is done at appropriate intermediary nodes, unnecessary information from each multicast group can be filtered out early, thus resulting in less total traffic.

Shah's, Ramzan's and Dendukur's objective is to minimize the total network bandwidth consumption. They describe an algorithm for an optimal filter placement in the IP multicast tree. CBM is different from application layer multicast in that IP multicast is enhanced by adding filters. The filters themselves might reside at the application or IP layer. The content filtering method will reduce network bandwidth usage and delivery delay, as well as the computation required at the sources and sinks. CBM reduces network bandwidth and recipient computation at the cost of increased computation in the network. The benefits of CBM depend critically upon how well filters are placed at interior nodes of the multicast tree.

2.4 Traditional Routing Algorithms

Traditional IP networks generally employ shortest path routing algorithms. The most commonly used routing algorithms on the Internet are the Dijkstra and the Bellman-Ford shortest path algorithms.

The Bellman-Ford algorithm [Bellman, 1958] solves the single-source shortest paths problem for a graph with both positive and negative edge weights. The algorithm maintains a distance value for each edge. At the beginning, it sets the source vertex distance to zero and all other vertices to a distance of infinity. It then loops through all edges in the graph and applies a relaxation operation to each edge. To guarantee that the distances between vertices have been reduced to the minimum, the relaxation process is repeated for n times where n is the number of vertices. The time complexity of the Bellman-Ford algorithm is $O(mn)$ where m is the number of edges and n is as defined above.

Dijkstra's algorithm [Cormen et al., 2001] is similar to the Bellman-Ford algorithm but has a lower execution time and requires non-negative edge weights. The cost of an edge can represent the distance between two associated vertices. For a given pair of vertices s and v, the algorithm finds the path from s to v with the lowest cost. The algorithm can also be used to build a shortest path tree from a source to multiple destinations by adding a new destination node with the minimum path cost from the source to the current tree at each step. Using the Fibonacci heap[1], Dijkstra's algorithm runs in $O(m + nlogn)$ time, where m is the number of edges and n is the number of vertices.

Another classical minimum spanning tree algorithm in graph theory is Prim's algorithm [Prim, 1957]. Prim's algorithm finds a minimum-cost spanning tree of an edge-weighted, connected, and undirected graph. The algorithm begins with a tree that contains only one vertex, it then repeatedly adds the lowest-weight edge that would connect a new vertex to the tree without forming a cycle. The time complexity of Prim's algorithm is the same as Dijkstra's algorithm if it is implemented using the Fibonacci heap. Both Prim's and Dijkstra's algorithms implement the greedy-choice strategy for a minimum spanning tree; the difference between them is the cost function used to add a new node to a current tree. In Prim's algorithm, a new node that has the minimum edge cost to a node already in the tree

[1] A Fibonacci heap is a heap data structure consisting of a collection of trees. Each tree satisfies the minimum-heap property, that is, the key of a child node is always greater than or equal to the key of the parent node [Levitin, 2007].

will be added, while in Dijkstra's algorithm, a node that has the minimum total cost to the source is added.

The three routing algorithms described above (Dijkstra, Prim and Bellman-Ford algorithms) can find optimal paths according to their routing metrics in polynomial time. However, they cannot determine routing paths based on multiple QoS constraints such as delay, delay jitter and bandwidth constraints, constraints which are required by many applications [Awduche et al., 2002].

2.5 Summary

In this chapter, the Web service architecture and its important standards were described. This architecture is important because it is the foundation of the work presented in this thesis. Traditional multicast protocols have also been explained to provide background for understanding the similarity-based multicast protocol for SOAP presented in the next chapter. In addition, common network routing algorithms such as Dijkstra's, Bellman-Ford's and Prim's algorithms have been reviewed. This revision creates a context for the work presented in Chapter 4 about a source network routing protocol that minimized SOAP network traffic.

Chapter 3

SOAP Binding BenchMark*

Hand-held devices with wireless capability are gaining popularity. SOAP is a text-based protocol for Web services, but it has high overhead, hence its suitability for resource-constrained devices over wireless networks needs to be reevaluated. Existing Web services often rely on HTTP — the most popular underlying transport protocol for SOAP messaging. While the HTTP protocol provides a number of benefits, including being able to pass through firewalls and being widely supported across different platforms, it was designed for wired networks with high bandwidth, low latency and low error rate transmissions. Due to the variability of wireless channels however, these assumptions do not hold in wireless environments. In this chapter, a benchmark of the performance of different underlying transport protocols for SOAP in wireless environments is reported. Through extensive testing, it is shown that SOAP-over-HTTP and SOAP-over-TCP are not well suited for wireless applications and lead to high latency and high transmission overhead. To overcome these limitations, the use of UDP as a binding protocol for SOAP is studied. The results obtained are promising and show that SOAP-over-UDP provides throughput that is up to six times higher than SOAP-over-HTTP in a wireless setting. Furthermore, using UDP to transport SOAP messages reduces transmission overhead by more than 30% compared to SOAP-over-HTTP. Finally, to illustrate where UDP binding can be useful, example applications are described.

3.1 Introduction

The mobile phone industry is enjoying an escalating growth all over the world. According to a recent market research [ADT, 2005], there are more than 300 million Java-enabled

* Preliminary versions of the work presented in this chapter have been published in [Lai et al., 2005] and [Phan et al., 2006a].

mobile handsets in use. In addition, there has been an explosive development of mobile telecommunications networks over the past few years. The expansion of 3G networks around the globe has changed the way people access services. Similarly, the sale of laptop and tablet type devices has also grown rapidly over the past ten years and these devices have become important tools for mobile workers who have to access and update data electronically [Mitchell, 2006].

Together with the expansion of mobile devices, their built-in capabilities are also being extended, especially towards programmable operating systems. This trend has led to the growth of new applications that can be built into these small devices. For example, the Compact .Net Framework [Microsoft Corporation, 2007] is supported on many Windows Mobile PDAs and smart phones, while J2ME [Sun Microsystems Inc., 2004a] is supported on many Palm devices and Symbian operating system-based phones.

In parallel, Web service paradigm has experienced a great surge of interest in both industry and academia. Web services and mobile technology have together influenced the design aspect of mobile services. Web services enable mobile devices to consume and provide services [Han et al., 2004]. There has been a significant amount of research into adapting mobile computing to the Web services architecture. Recently, Sun Microsystems released JSR 172 [Sun Microsystems Inc., 2004b], a specification that addresses the use of XML, SOA and Web service on J2ME devices.

However, SOAP was originally designed for wired networks; it is poor at dealing with the challenges of wireless communications and the resource limitations of mobile devices — slow CPU, low memory and limited battery life [Chen and Nath, 2004]. Currently, SOAP performance is one of the critical integration issues attracting a lot of research in the area of mobile Web services. Other issues are context-awareness [Han et al., 2004], adaptability and security [Chen et al., 2005].

Existing research on SOAP performance [Davis and Zhang, 2002; Devaram and Andresen, 2002; Ng et al., 2003; Tian et al., 2004] has found that current implementations of SOAP using HTTP as the transport protocol are slower than other middleware technologies such as Java RMI and CORBA. In particular, Chiu et al. [2002] investigated the limitations of SOAP for scientific computing. Their experiments compared SOAP to Java RMI by sending large arrays of doubles; the results showed that SOAP is about ten times slower than Java RMI. Kohlhoff and Steele [2003] studied SOAP performance in business applications in the context of trading systems. Their study compared SOAP to a binary wire format, called Common Data Representation (CDR), which is used in CORBA communication. Their results showed that SOAP is 2–3 times slower than the Internet Inter-ORB Protocol (IIOP) [OMG, 2007] that uses CDR.

Despite SOAP performance issues, XML, SOAP and WSDL together provide a framework for data exchange across various computing platforms and environments. Thus, Web services are used in favor of other middleware technologies for interoperability between heterogeneous systems. The binding implementation of SOAP over HTTP is universally used on the Internet today, but this implementation has some drawbacks especially in mobile environments. The disadvantages of using SOAP over HTTP in mobile computing are mostly due to the nature of HTTP and TCP protocols themselves. When a SOAP message is sent over HTTP, the following overhead results from the connection-oriented property of TCP:

- TCP requires a connection to be established before any data can be transmitted. Moreover, as data is received, acknowledgement packets are sent. This leads to additional overhead which may not be justifiable where bandwidth and client power are limited and reliable transmission of packets is not required.
- SOAP messages that carry only a small amount of data can finish transmitting while the TCP connection is in its slow start phase. This results in poor utilization of the available bandwidth. The problem is particularly severe in wireless environments due to high round trip time.
- The congestion avoidance mechanism in TCP assumes packet losses are always due to congestion. However, in a wireless network, packet losses are usually due to disconnections and transmission errors. This impacts on bandwidth utilization.

This chapter aims to provide a detailed comparison between three different SOAP binding options, SOAP over HTTP (with TCP as transport protocol), SOAP over TCP and SOAP over UDP in a wireless environment. These comparisons are important because the results obtained will highlight the strengths and weaknesses of each binding option. With this knowledge, the most suitable binding can be determined for different types of usage scenarios. The results from this study will be useful for other researchers to examine the effectiveness in performance of these transport protocols to the development of Web service in wireless networks. Experimental results show that SOAP-over-UDP provides performance benefits over the traditional SOAP-over-HTTP binding.

The rest of this chapter is organized as follows. The next section examines the limitations of mobile devices and wireless networks, different SOAP engines used in wireless environments and the SOAP binding framework. Related work on mobile Web service performance is presented in Section 3.3. This is followed by a section outlining the experimental results and analysis of the benchmark. Section 3.6 presents some sample applications that are suitable for different kinds of SOAP bindings. Finally, this chapter concludes by a summary section reviewing the advantages of UDP binding over HTTP and TCP bindings.

3.2 Background

In this section, background on concepts related to the work presented in this chapter is described. Firstly, an overview of mobile Web services are explained, followed by a discussion of limitations of mobile devices and wireless networks. Subsequently, some popular SOAP implementations for mobile devices are described. Finally, common SOAP bindings with HTTP, TCP and SMTP transport protocols are presented.

3.2.1 Mobile Web Services

With the advancement in wireless technologies in general, and mobile device capabilities in particular, ubiquitous access of mobile Web services continues to be an important research topic. Mobile Web services are the deployment of Web services in mobile environments. Web service clients are not limited to desktop computers but extended to mobile devices such as laptops, tablet PCs, PDAs, smart-phones and mobile phones. However, the evolution of mobile Web services is not only about requesting services from mobile clients but also providing services on mobile devices. In the general case, a mobile device can act as either a Web service requestor or provider or both.

Due to the nature of mobile wireless networks, existing middleware infrastructure cannot efficiently support Web services on mobile devices. Deploying a SOAP client on a mobile phone might reveal some performance costs related to slow transmission and processing of both HTTP commands and XML. The mobility of devices raises a number of issues, such as service discovery, networking and security that are different from those in the domain of fixed workstations. Additionally, the constraints on computational power and battery life on mobile devices add another dimension to the challenge.

3.2.2 Limitations of Mobile Devices and Wireless Networks

Mobile devices generally have much lower CPU power, storage capacity and smaller memory than servers and desktop computers. Therefore, Web service implementations on mobile devices should take computational time, memory and energy resources into careful consideration.

Mobile devices are often dependent on wireless communication protocols for connectivity. Some wireless connections such as 802.11 or Bluetooth are short in range. The users of mobile devices may also experience frequent disconnections due to being out of range of a network access point — services that are offered on mobile devices should be tolerant to such network failures because the services may not be accessible all the time. In addition, mobile devices' constrained energy supply places a restriction on the design of mobile Web services that services should not consume too much device energy.

A brief summary of issues that are inherent to wireless environments is provided below.

- **Intermittent connections**: Mobile devices do not have network connectivity everywhere at all times. Whether it is a cellular, 802.11 network or Bluetooth, wireless networks suffer from regular disconnections.
- **Limited bandwidth**: Even though the available bandwidth of wireless networks is increasing, many wireless networks have limited bandwidth of around 64–300 Kb/s.
- **Latency**: Wireless network connections are slow; thus high latencies not only affect applications but may frustrate users.
- **High cost**: In spite of the dropping trend, wireless rates are expensive.

3.2.3 SOAP Implementations for Mobile Devices

A number of SOAP implementations such as as gSOAP [Angelen, 2003], kSOAP [Haustein and Seigel, 2003] and pocketSOAP [Fell, 2004] have been implemented on mobile devices. Among the three, kSOAP was chosen as the SOAP client implementation in the SOAP binding benchmark presented in this chapter, because of its popularity and ease of use. In addition, kSOAP is written in Java and designed to run in an MIDP (Mobile Information Device Profile) [Sun Microsystems Inc., 2007] environment which most mobile phones support. The SOAP API of kSOAP provides a small footprint implementation of XML aimed at developing applications for mobile devices using J2ME [Sun Microsystems Inc., 2004a]. The implementation of kSOAP is based on kXML [Haustein, 2005], an open source API that provides an XML pull parser especially designed for constrained environments such as Applets, Personal Java and MIDP devices.

3.2.4 SOAP Bindings

A SOAP binding is the enveloping of a SOAP data payload within underlying protocols. Such bindings have to specify the rules for encapsulating or decapsulating SOAP data from the underlying protocol. This section introduces some popular SOAP bindings.

SOAP-over-HTTP

Over the Internet, HTTP is the protocol that is most widely used for SOAP binding. Because HTTP is one of the core protocols of the Internet and is widely supported by Web servers, SOAP-over-HTTP is the only concrete binding specification defined in the SOAP binding framework proposal [Gudgin et al., 2007]. Because it is often allowed to pass through firewalls, it is a convenient candidate for transporting SOAP. The recommended version of HTTP for SOAP binding is HTTP/1.1 and all SOAP implementations provide this binding.

A SOAP message can be transported using HTTP by encapsulating the SOAP request into the message body of a HTTP GET or HTTP POST. Similarly, a SOAP response can be encapsulated into the body of a HTTP response [Mitra and et al., 2007]. HTTP binding provides reliable message transport, flow and congestion control. However, SOAP-over-HTTP has some drawbacks, for instance it does not support peer-to-peer messaging exchange between SOAP nodes. The response time is generally higher when using HTTP as the transport protocol for SOAP because of the handshake process occurring at the TCP layer.

SOAP-over-TCP

In order to transport a SOAP message using TCP as a direct underlying protocol, a SOAP message is stored in the data octets part of a TCP packet (see Figure 3.1). There is not yet any official specification for SOAP binding with TCP, however, Apache Axis [Apache Software Foundation, 2007a] and Microsoft Web Service Enhancement (WSE) 2.0 [Microsoft Corporation, 2006] include APIs that enable the sending of SOAP messages via TCP. Two types of TCP-based messaging (namely synchronous and asynchronous) are supported in the WSE 2.0. In synchronous TCP messaging, the request and response messages are exchanged over a common TCP connection as in the Request-Response MEP. With the asynchronous mode, the TCP connection is removed after a message has been sent, representing the one-way messaging pattern in SOAP.

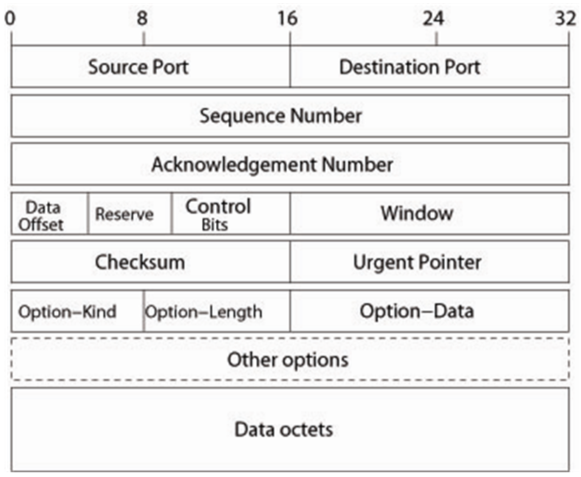


Figure 3.1: A TCP packet

SOAP-over-SMTP

In addition to the HTTP binding, a SOAP-over-email binding is also present in the W3C specification [Mitra and et al., 2007]. However, unlike the HTTP binding, which forms part of the SOAP standard, the SOAP-over-email binding is only presented in the specification as an example to demonstrate the realization of the SOAP binding framework. In the SOAP-over-email binding, SOAP messages are piggy-backed on SMTP packets.

3.3 Related Work

Existing studies on improving SOAP performance in wired and wireless environments are discussed in this section.

SOAP performance has been the main focus of many research papers over the past few years. These studies have proposed several approaches for improving the overall performance of SOAP to make Web services technologies more viable in replacing traditional distributed object technologies such as CORBA and Java RMI. Gryazin and Seppala compared SOAP with CORBA for mobile devices and found that the performance of Web services does not depend on the implementation of SOAP [Gryazin and Seppala, 2004]. They also concluded that SOAP is more suitable for larger scale systems with non-critical architectures while CORBA is more robust and flexible for mobile applications.

Head et al. [2005] proposed a standard benchmark suite for different SOAP implementations. They reported on the performance of existing popular SOAP implementations including Axis Java [Apache Software Foundation, 2007a], gSOAP [Angelen, 2003] and bSOAP [Abu-Ghazaleh et al., 2004], from aspects of serialization, deserialization, namespace and latency. Their results show that gSOAP is the best for latency critical applications. Axis Java does not perform well either in Windows or Linux environments. Protocol gSOAP also scales well with an increase in the size of complex data types, and can reduce the Web service overhead. In contrast, Axis Java does not provide good scalability. Protocol bSOAP is comparable to gSOAP when sending arrays of doubles, integers and strings.

The performance of Web services using J2ME and kSOAP clients measured on a PDA and an IBM Thinkpad laptop was reported by Bansal and Dalton [2002]. These clients access a temperature service and a translation service which are available from the Internet. Bansal's and Dalton's results show that the main bottlenecks are low available bandwidth causing large transmission time and the high cost associated with XML parsing.

Guido and Ralf [2004] proposed a framework to allow accessing Web services from J2ME mobile devices using WAP. Because of the high overhead of HTTP, the authors develop a Java Wireless Session Protocol (WSP) implementation to avoid the TCP three way handshake and to reduce header size. Their experiments show that the protocol header can be reduced by more than a third by choosing WSP instead of HTTP to transport SOAP messages. However, the memory footprint (the size of the MIDlet[1]) required when using WSP (89K) is higher than when using HTTP (54K). The time spent on parsing SOAP messages was similar for HTTP and WSP, however the transmission time of the SOAP messages was reduced by about a third by choosing WSP instead of HTTP.

Researchers at the DoCoMo USA labs proposed a set of SOAP optimization techniques called Wireless SOAP (WSOAP) to transmit SOAP messages across wireless networks [Apte et al., 2005] with significant bandwidth reduction compared to generic SOAP. WSOAP's principles are based on adaptive encoding which relies on the underlying WSDL service description being known by both the sender and receiver. Their experiments demonstrate that WSOAP can reduce the size of SOAP messages by 3–12 times compared to generic SOAP. Their proposed optimization techniques outperform the differential encoding technique proposed by Werner et al. [2004] and WAP binary XML [W3C, 1999] by large factors.

Lee and Fox [2004] discussed issues and strategies to extend the reliability of Web services in wireless communication environment. A Web Service-Wireless Reliable Messaging (WS-WRM) framework is designed to interoperate with other reliability schemes such as WS-Reliability [OASIS, 2004] and WS-ReliableMessaging [Bilorusets et al., 2005] which are

[1] A MIDlet is a Java program for embedded devices

general specifications for Web service reliable messaging. WS-WRM uses negative acknowledgement messages to reduce the data communication overhead. The limited data storage capability on the mobile end-points is also taken into consideration. Not all three messaging patterns of HTTP binding (Request-Response, Callback and Polling) [OASIS, 2004] are available on mobile devices. Due to this limitation, Lee and Fox implemented WS-WRM on top of their NaradaBrokering service [Fox and Pallickara, 2002] instead of binding WS-WRM directly to HTTP. NaradaBrokering service, which was built by the same authors, is a general purpose event brokering system designed to run a large network of cooperating broker nodes. The authors also propose a message integrity feature by offering multiple algorithms, and the services can specify the algorithm based on their requirements.

3.4 SOAP-over-UDP

UDP is a simple, low-overhead transport protocol, Figure 3.2 shows a UDP datagram. Unlike TCP, it does not provide any flow-control mechanism and only guarantees best-effort delivery of packets. Packets delivered by UDP may be duplicated, arrive out of sequence or not even reach their destination at all. However, due to its simplicity, UDP provides a number of benefits over TCP:

- UDP does not require a connection to be established before sending a packet. Each UDP datagram carries its own destination address and is routed independently of other packets. This reduces the setup time associated with sending a message.
- UDP packets are smaller than TCP packets. The UDP header is only eight bytes in length, in comparison to the TCP header which is at least 20 bytes in length. For wireless environments where bandwidth availability is low and users are charged by the amount of data they transmit and receive, UDP provides a cheaper alternative to TCP.
- UDP supports multicasting. This opens up the opportunity to create push-based and publish/subscribe Web services, where SOAP messages or notifications are sent to multiple clients periodically or triggered by an event.

Because of these advantages, SOAP-over-UDP as an alternative binding for SOAP messaging is examined. UDP is light-weight, unreliable and connectionless. Due to its simplicity, UDP has little overhead and offers fast packet delivery. These characteristics may be desirable for applications where high transfer rate is valued over reliability, for example, real-time

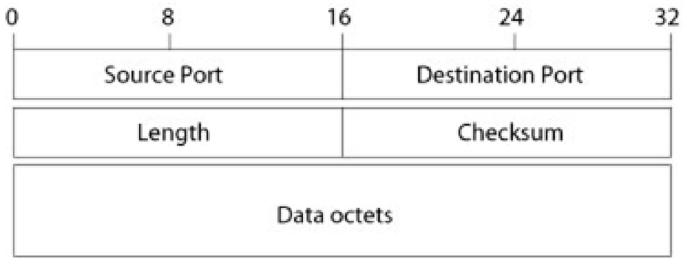


Figure 3.2: A UDP datagram

multimedia streaming. UDP is also suitable for quick exchange of information that is small in size. An example of this is the domain name service (DNS), which uses UDP datagrams to transmit domain name lookup requests [Ross and Kurose, 2000].

The recently released SOAP-over-UDP specification [BEA Systems Inc. et al., 2004] defines how a SOAP message can be encapsulated into the *data octets* part of a single UDP packet (see Figure 3.2). The SOAP message must be small enough to fit within a single datagram, less than 65,536 (2^{16}) bytes in size. The specification supports four message exchange patterns (MEP) as follows:

- Unicast one-way: This is a pattern for the exchange of a non-SOAP message acting as a request followed by a SOAP message acting as a response.
- Multicast one-way: Only the request is sent out to a multicast or broadcast address.
- Unicast request, unicast response: The SOAP Request-Response MEP defines a pattern for the exchange of a SOAP message acting as a request followed by a message acting as a response.
- Multicast request, unicast response: The response must not be multicast. The sender of the request might receive multiple responses. The correlation between the request and response messages is realized by using Web Services Addressing properties.

3.5 SOAP Binding Benchmark

This section presents the different experimental setups and results. The experiment was intended to identify the performance tradeoffs when selecting different transport protocols for SOAP. Three SOAP bindings (SOAP-over-HTTP, SOAP-over-TCP and SOAP-over-UDP) were tested and compared to each other. There are three sets of tests. In the first set, three transport protocols were tested in a loopback mode. In the second set, the tests were carried out between two computers under a WLAN mode. The last set of tests involved the deployment of SOAP clients on hand-held devices.

3.5.1 Experimental Setup

A test system was set up to measure the performance of the three transport protocols for SOAP communication. The test driver used for the benchmark is a modified version of the WSTest 1.0 [Sun Microsystems Inc., 2004c] developed by Sun Microsystems. WSTest was specified as part of a benchmarking experiment comparing the performance of J2EE Web services and .Net Web services. Sun's WSTest has been revised to include the following Web service calls [2]:

- echoVoid: sends and receives an empty message, no deserialization or serialization of payload is required. This service call examines the performance of the transport protocol infrastructure.
- echoDouble: sends and receives a message of type *Double*.
- echoString: sends and receives a message of type *String*.
- echoStruct: sends and receives an array of 10 elements, with each element consisting of a structure. The repeating structures within the array contain one element each of type *Integer*, *Float* and *String* data types.
- echoList: sends and receives a linked list of 10 elements, each element consists of the same structure used in Struct as defined above.

These Web service calls were selected to separate message headers from payload and to create a test-bed with different message types and sizes.

The purpose of these tests is to examine SOAP performance with different data sizes and types. Each Web service method is an echo function in which a client sends a data message of a particular data structure to the server and the server responds with the same message. The test is highly configurable. The number of concurrent running clients, the startup (time allocated for warm up of the system, running (the interval when throughput is measured) and ramp-down (time allocated for completing operations) times can be specified. The size of the data payload in each service call can also be varied.

[2] Web service calls: *echoDouble* and *echoString* were added from the original WSTest driver.

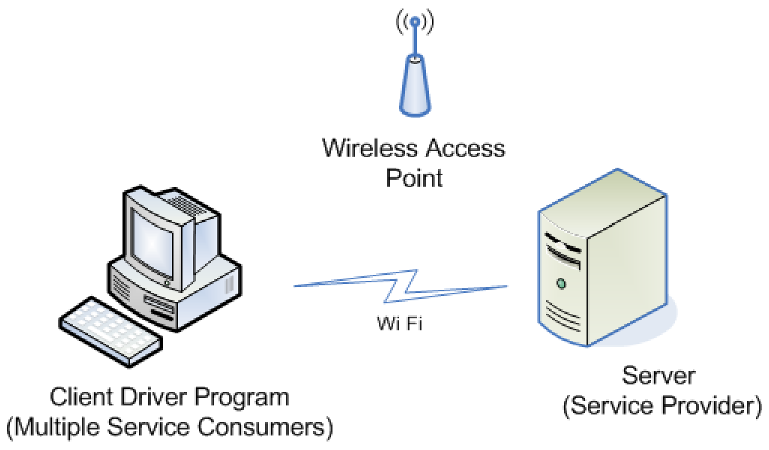

Wireless Access Point
Wi Fi
Client Driver Program (Multiple Service Consumers)
Server (Service Provider)

Figure 3.3: Experiment setup in WLAN mode

The WSTest service was hosted on a PC server while multiple clients were simulated on a PC client to request the service over a wireless network. The machines used for testing are two desktop PCs having similar hardware configurations: Pentium III, 966Mhz CPU with 256 Mb RAM, running Java 2 SDK 1.4.1 on a Redhat Linux 9 operating system. Both machines are equipped with an 802.11b wireless antenna each and connected via a Netgear WG602 wireless router. The experiments were performed using a wireless connection speed of 54Mbps. The SOAP frameworks used in the experiments were Apache Axis 1.2 [Apache Software Foundation, 2007a] on the server and kSOAP 2.0 [Haustein and Seigel, 2003] on the client. Tomcat 5.0 [Apache Software Foundation, 2007b] was used on the server to host the Web service. Both the client and the server were written in Java. The testing environment is explained in Figure 3.3.

Performance was tested under three modes:

- **Loopback**: The server and the client both ran on the same computer. Connection was via the loopback (localhost) interface, therefore no packet loss or bandwidth limits apply to this mode.

- **WLAN**: Computer 1 hosted the server and computer 2 ran the client application that generated multiple concurrent requests to the Web service. The two computers were connected via a wireless router. They were the only machines connected to the router at the time of the test. A series of pings from computer 2 to computer 1 showed that the average round trip time was 27 ms.

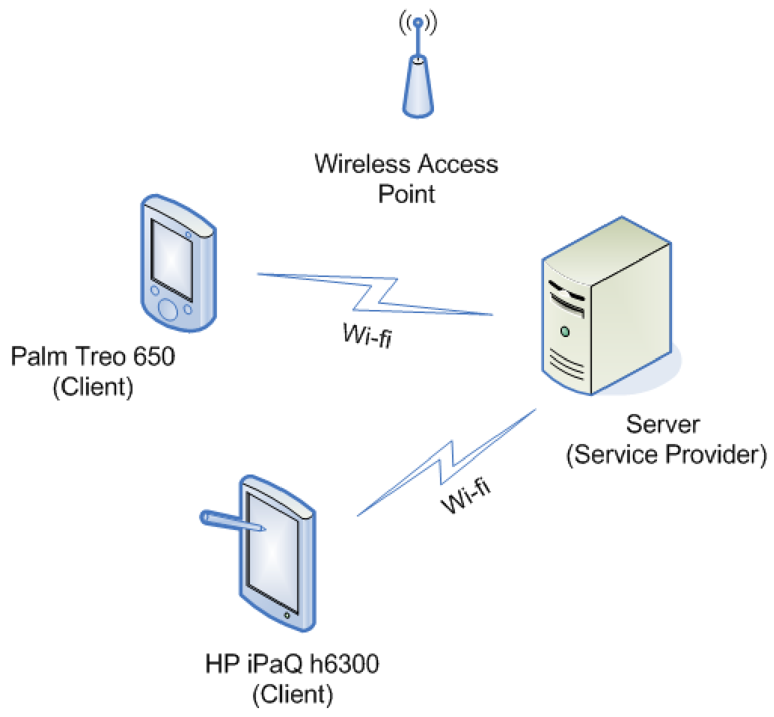


Figure 3.4: Experiment setup in mobile device mode

- **Mobile device**: The Web service was hosted on the server. Client requests were made from mobile devices. The client tests were deployed on 2 PDAs: 1 HP iPAQ h6300 series and 1 Palm Treo 650. Two mobile clients accessed the service via a WLAN connection at the same time, and the response time of each request type was reported. The *echoDouble* call was not tested since the mobile devices did not support floating point. For the same reason, the Struct and the List data types do not include the float elements. Figure 3.4 illustrates the test environment for this mode.

The performance metrics used as part of the performance testing include:

- Throughput: The average number of Web service operations executed per second. A Web service operation here corresponds to a request/response cycle.

- Average response time: Average time taken to process a request at the server and send a response back to a client.

3.5.2 Experimental Results and Analysis

This section summarizes the performance tests. First, the protocol overhead of each binding is analyzed. Then experimental results for loopback, WLAN and mobile device modes are presented.

Transmission Overhead

In this section, the metrics used consist of packet overhead which is the sum of the SOAP payload and the protocol header; and connection overhead which is the sum of generated traffic as the result of connection establishment.

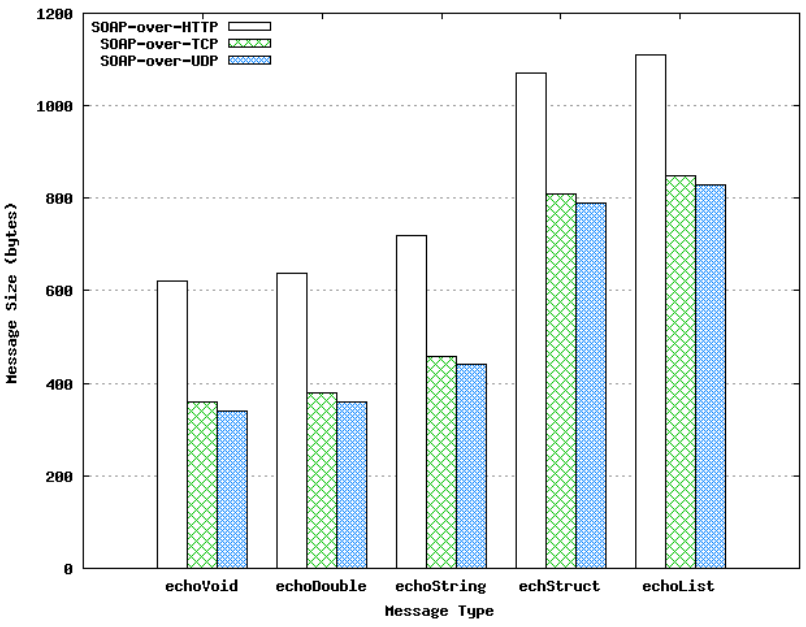


Figure 3.5: Bytes sent for different message types

Figure 3.5 shows the size of different messages created by different binding options. The messages shown here are request messages for different Web service operations. Since all service operations are echoing functions, the size of a response is similar to its request. It is shown that for an empty payload message, echoVoid, SOAP-over-HTTP transmits nearly twice the number of bytes compared to SOAP-over-UDP. With small size messages such as echoDouble and echoString, clients only need to send to the server a parameter of type Double or of type String (under 50 bytes), however the HTTP and TCP headers add significant overhead to the SOAP request for SOAP-over-HTTP messages. UDP has some header overhead, however this is only 8 bytes. Similar behaviors are seen for the echoStruct and echoList message types.

The size of echoStruct and echoList requests are significantly larger than the other message types because they contain an array of simple data types or a linked list of struct data types. With larger message types, SOAP-over-HTTP adds about 30% to the size of a SOAP message, compared to SOAP-over-UDP. In all cases, there is no significant difference in message size between TCP and UDP bindings because the TCP header is also small (24 bytes). For messages carrying only a few parameters (e.g. echoVoid, echoDouble and echoString), using UDP can reduce message size by about 50% compared to HTTP. This is significant for mobile clients that repeatedly send simple SOAP requests to a server, because such a reduction in message size leads to a huge saving in transmission cost and client energy consumption.

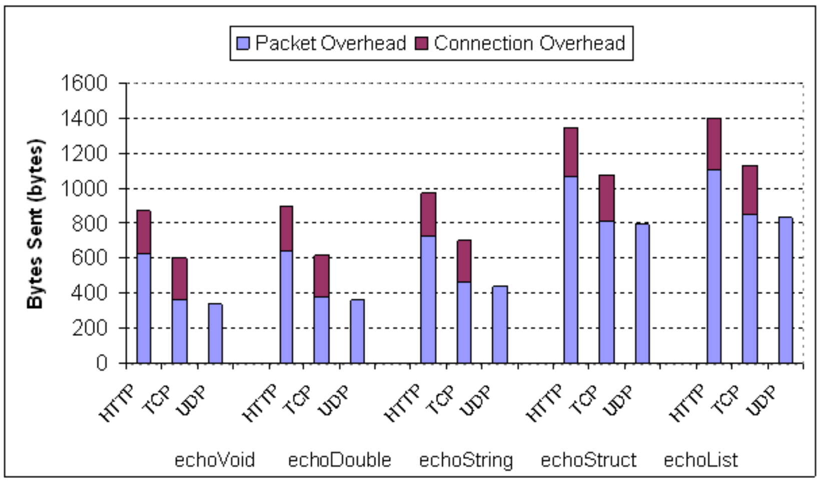


Figure 3.6: Connection overhead versus Packet overhead for different bindings

Figure 3.6 shows a breakdown of the message overhead in terms of packet overhead and connection overhead using HTTP, TCP and UDP bindings for the different message types. For example, SOAP-over-HTTP messages contains a HTTP header, a TCP header and the actual SOAP message. It can be seen that TCP and UDP have very similar packet overheads, however the connection overhead of using SOAP-over-TCP makes TCP more expensive than UDP. In the case of request messages, the connection overhead is a result of the connection establishment handshake consisting of three separate packets (SYN, SYN-ACK and ACK)

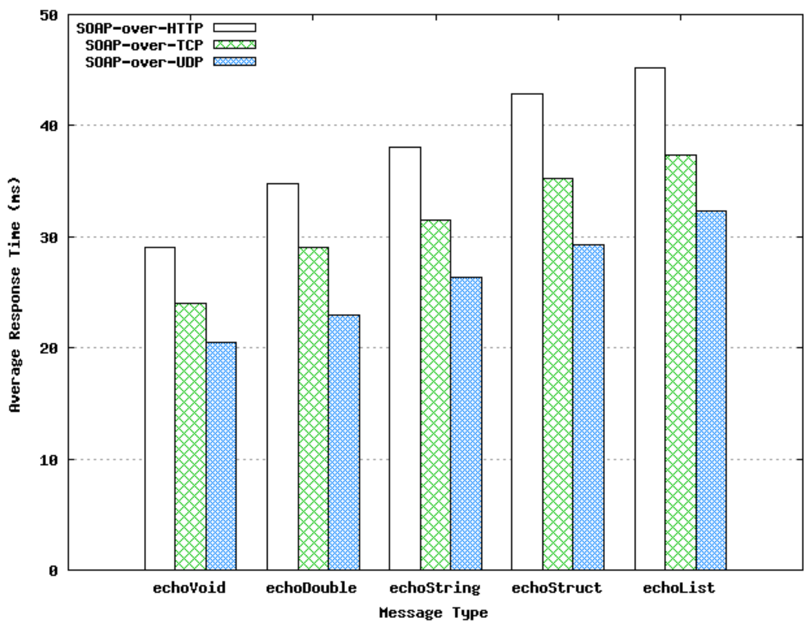


Figure 3.7: Average response time for a test with 50 concurrent clients (loopback)

[Allman et al., 1999]. Once the server has responded to the requests, four more packets (FIN and ACK each from sender and receiver) are used to tear down the connection. The connection overheads of echoStruct and echoList are slightly increased further as the SOAP responses for these requests do not fit into a single TCP packet. As a result, a large response has to be segmented into multiple packets, which introduces TCP acknowledgement packets with additional overheads into the message exchange.

Experiments performed over loopback connection

Next, the impacts of using different binding options on execution time and throughput are studied in loopback mode. Figure 3.7 shows the average time it takes for each client to execute a request over the loopback connection when there were 50 simulated clients concurrently making requests. For various message types, SOAP-over-UDP is significantly faster than SOAP-over-TCP and SOAP-over-HTTP. The poor performance of TCP and HTTP is mainly due to the additional cost in encoding and decoding HTTP and TCP headers and also the time spent on connection establishment. SOAP-over-TCP is roughly 17% faster than SOAP-over-HTTP, while SOAP-over-UDP is around 30% faster than SOAP-over-HTTP.

Another important performance metric is throughput, reflecting the scalability of the

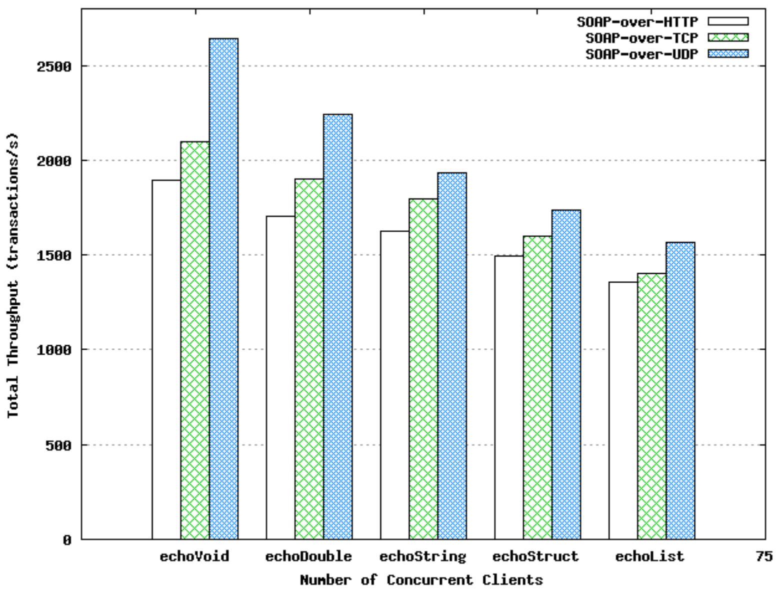


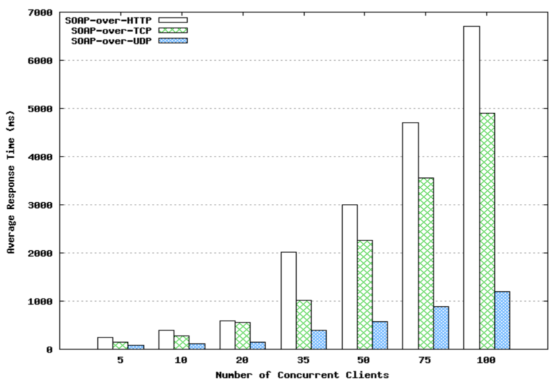
Figure 3.8: Total throughput for a test with 50 concurrent clients (loopback)

binding options tested. Figure 3.8 plots the throughput (number of messages processed per second by the server) under the loopback connection scenario. With small message types (echoVoid, echoDouble and echoString), it is found that the throughput of SOAP-over-HTTP (1800 transactions per second) is slightly smaller than that of SOAP-over-TCP (2000 transactions per second). The throughput of SOAP-over-UDP is significantly higher (over 2500 transactions per second) than both SOAP-over-HTTP and SOAP-over-TCP. Using the UDP binding, SOAP messages can be sent without setting up a connection, thus minimal system resources are used. Furthermore, the simplicity of the UDP protocol means the server can process UDP headers faster than TCP and HTTP headers.

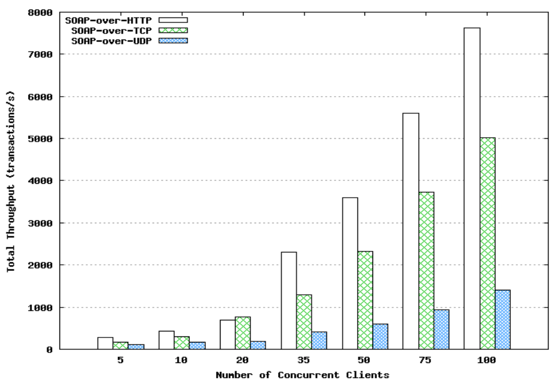
With echoStruct and echoList message types, the sizes of the messages are much larger than the protocol headers; therefore, the advantage of light UDP messages does not produce a significant improvement in the total throughput. The total throughputs of HTTP, TCP and UDP bindings with echoList message type are 1350, 1400 and 1550 transactions per second respectively.

Experiments performed over Wi-Fi connections

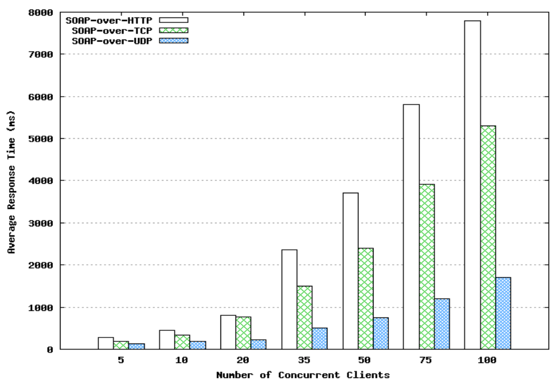
The execution time and throughput experiments were repeated over a 802.11b Wi-Fi connection using computer 1 and computer 2. The computers were connected to a wireless router



(a) echoVoid message type



(b) echoDouble message type



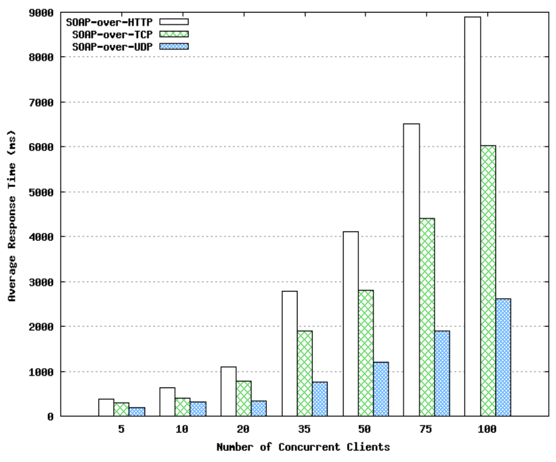
(c) echoString message type

Figure 3.9: Average response times of different SOAP bindings for echoVoid, echoString and echoDouble message types under different numbers of concurrent clients

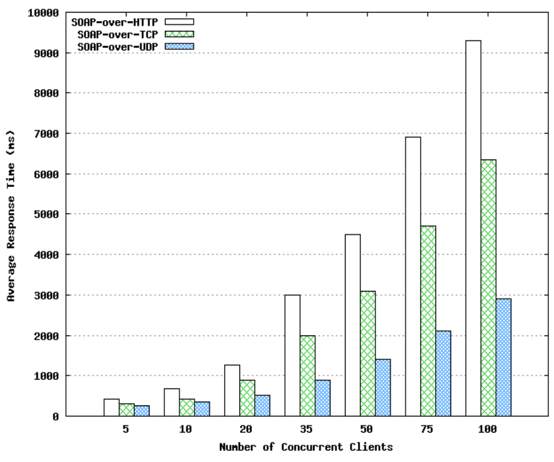
at 54Mbps. These experiments were carried out for various numbers of concurrent clients.

The average response times of different SOAP bindings for the echoVoid, echoString and echoDouble message types are plotted in Figure 3.9. The graphs show a significant difference between the result in Wi-Fi connection (see results for 50 clients) compared to those collected from the loopback experiments. For SOAP-over-HTTP and SOAP-over-TCP, the average response time over the wireless network is about 100 times higher than over loopback. The difference with the UDP results is less, average execution time over wireless is around 30 times higher than over loopback.

These results show that SOAP-over-UDP is much faster than HTTP and TCP, for a



(a) echoStruct message type



(b) echoList message type

Figure 3.10: Average response times of different SOAP bindings for echoStruct and echoList message types under different numbers of concurrent clients

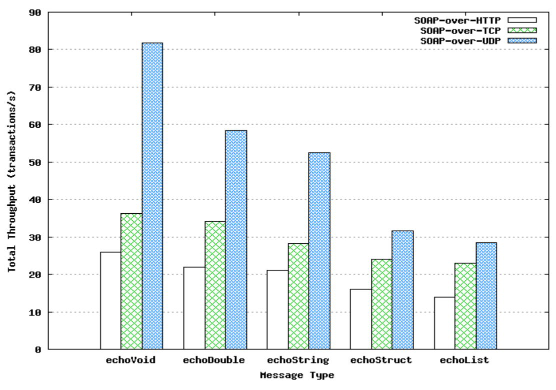
number of reasons. Firstly, in the HTTP and TCP bindings, connection establishment, packet acknowledgement, and connection tear-down require more packets. Due to the high round trip time in a wireless network, the higher number of packets exchanged leads to an increase in execution time. Secondly, unlike the loopback connection (which is reliable), the wireless link is susceptible to packet losses. When packet loss occurs, the retransmission mechanism in TCP is triggered, leading to an increase in execution time. Finally, when an acknowledgement packet is lost, the TCP client must wait for the TCP retransmission timeout (RTO), leading to a further increase in execution time. UDP does not spend any time in establishing and tearing down connections, therefore a large number of packets can be transmitted quickly.

Due to the unreliable nature of the wireless link, some packet loss occurred in the UDP binding case (below 1% for small messages and small number of concurrent clients, and around 3–5% for large messages and high number of simultaneous requests). However, for applications with high performance and low reliability requirements, SOAP-over-UDP provides significantly better performance than SOAP-over-HTTP and SOAP-over-TCP. The results show that for small to medium messages (echoVoid, echoDouble and echoString) and a large number of client requests (over 35 clients) SOAP-over-UDP is about 4–5 times faster than HTTP, and about 3–3.5 times faster than TCP. The results are less significant for smaller numbers of clients (under 35 clients). In particular, SOAP-over-UDP is only approximately 2–3 times faster than HTTP and TCP. SOAP-over-UDP scales quite well with a large number of concurrent clients. The results for SOAP-over-HTTP and SOAP-over-TCP are quite similar for small networks, however SOAP-over-TCP surpassed SOAP-over-HTTP by about 30% for larger networks.

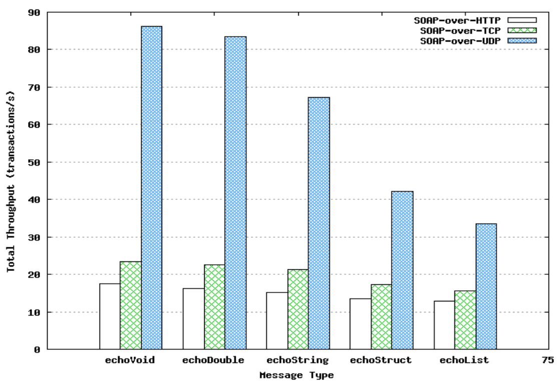
The average response times for the echoStruct and echoList message types are shown in Figure 3.10. For larger messages (echoStruct and echoList), the difference in performance between SOAP-over-UDP and SOAP-over-HTTP is slightly less than in the experiments with smaller messages which are shown in Figure 3.9. Specifically, the UDP binding outperforms the HTTP binding by about only 2 times for a network of 10 clients and about 3 times for a network of 100 clients in the experiments with echoStruct and echoList message types. The performance differences between the HTTP binding and the TCP binding do not vary much for different sizes of messages (the TCP binding is still around 30% faster than the HTTP binding with echoStruct and echoList message types).

The total throughputs of different bindings over the Wi-Fi network for the scenario of 10 clients are shown in Figure 3.11(a). As expected from the average response time results, SOAP-over-HTTP and SOAP-over-TCP provide significantly lower throughput compared to SOAP-over-UDP. For small messages (echoVoid and echoDouble), SOAP-over-UDP achieves throughput nearly 3 times higher than SOAP-over-HTTP, while for large messages (echoString, echoStruct and echoList), the throughput under the UDP binding is 2 times higher. The performance difference is less for larger messages because TCP and HTTP perform better when transmitting large amount of data because for large data transfers, the connection management overhead of TCP and HTTP becomes less significant compared to the transmission cost of the actual payload.

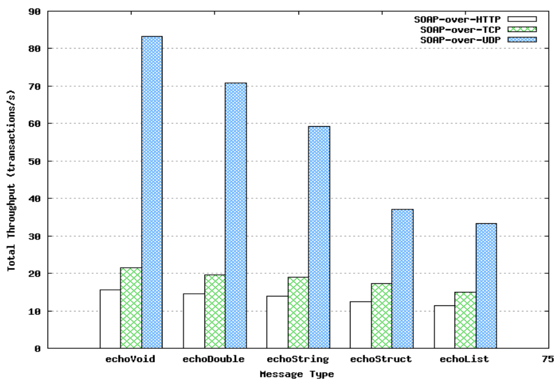
Figure 3.11(b) illustrates the throughput in the scenario of 50 clients. The performance differences between SOAP-over-HTTP and SOAP-over-UDP in this scenario are larger than the results in the case of 10 clients. Throughput achieved using the UDP binding is approximately 5–6 times higher than the HTTP and TCP bindings for small messages (echoVoid, echoString and echoDouble) and 2–3 times higher for large messages (echoStruct and echoList). There is no noticeable difference in the throughput performance between the HTTP and TCP binding options. Similar results were obtained for the experiments with 100 concurrent clients (see Figure 3.11(c)). The throughput performance of SOAP-over-UDP scales well as the number of clients in a network increases.



(a) 10 concurrent clients



(b) 50 concurrent clients



(c) 100 concurrent clients

Figure 3.11: Total throughputs of different SOAP bindings under test scenarios of 10, 50 and 100 concurrent clients

Experiments performed on mobile devices over Wi-Fi connections

This section reports the results obtained from the experiment carried out on mobile devices over the wireless network. The charts in Figures 3.12 and 3.13 show the average response time and the total throughput observed during this experiment. Although the performance of all three SOAP bindings under the mobile device tests is considerably lower than that under the desktop computer tests, the performance variances between the UDP binding and the HTTP binding are similar. Due to limited computational power of the mobile devices, the serialisation and the deserialisation of SOAP request and response messages took much longer. It took almost 5s for the devices to receive an echoList message using the HTTP

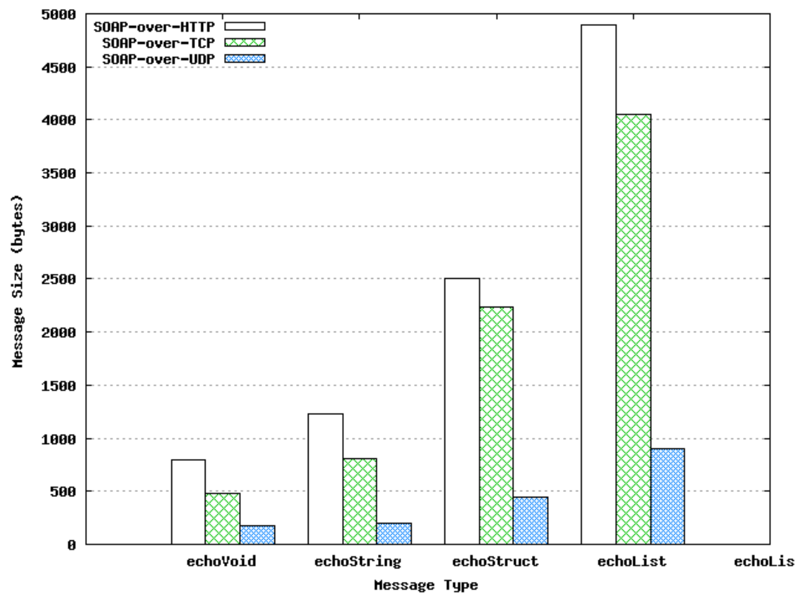


Figure 3.12: Average Response time for a test with 2 mobile clients

binding but it only took 0.5s in the same test carried out on a computer. It is observed that the response time increases sharply as the size of the messages increases for all types of binding options. This indicates that mobile Web services do not cope well with large messages.

Similar trends were observed in the total throughput results. With the echoList message type, the throughput of the UDP binding decreases to nearly one fifth of that with the echoVoid message type (a reduction from 11.6 transactions/s to 2.5 transactions/s). However, the total throughput of the UDP binding is still significantly higher than that of the HTTP and TCP bindings. Specifically, with the echoStruct message call, SOAP-over-UDP achieves 4.5 transactions/s throughput while both SOAP-over-TCP and SOAP-over-UDP achieve less than 1 transation/s throughput.

3.6 Application

This section gives examples of different applications that are suitable for the various SOAP bindings.

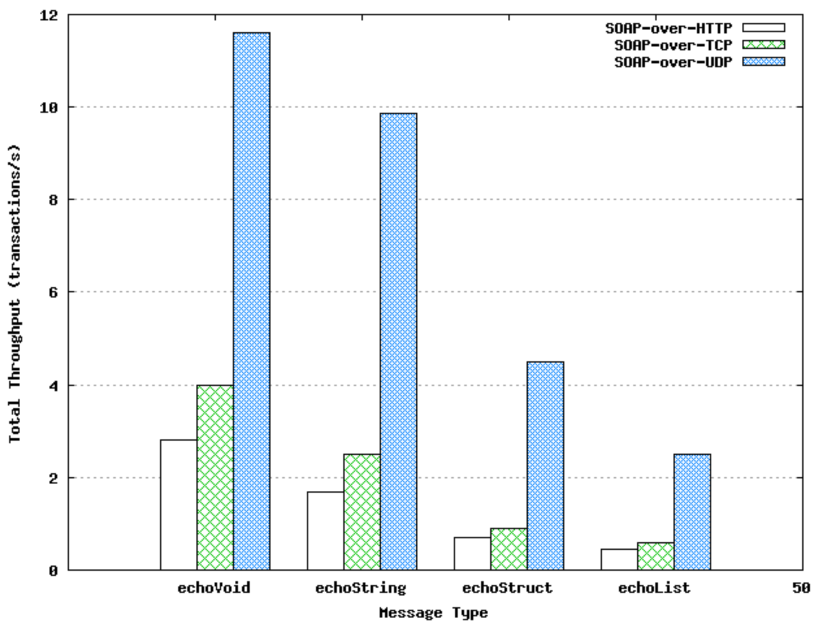


Figure 3.13: Total throughput for a test with 2 mobile clients

3.6.1 SOAP-over-HTTP

A significant advantage of the HTTP binding is that it allows SOAP messages to pass through firewalls and be processed by web servers. However, due to the size of the HTTP header and the use of TCP as the underlying transport protocol, SOAP-over-HTTP incurs relatively high overhead for small SOAP messages. HTTP binding is still a good choice for applications that require high level of reliability or high data payload. Some examples where SOAP-over-HTTP is suitable include:

- *Translation service* - where clients provide text in one language and the Web service performs translation to another language and sends it back to the client.
- *Payment service* - where the Web service collects sensitive credit card or banking information from clients, processes the transaction and replies with a receipt/confirmation. HTTPS can be used instead of HTTP to provide security.

3.6.2 SOAP-over-TCP

The experimental results show that SOAP-over-TCP does not provide significant benefit over SOAP-over-HTTP in terms of throughput and execution time. However, the packet size of a simple SOAP request using TCP is around 30% smaller than the equivalent request using the HTTP binding. As a result, TCP is suited to applications where clients send short requests to the server to request large amount of data, and the data needs to be transported reliably. An example of such an application may be:

- *Scientific data transfer* - where clients sent requests to retrieve large scientific data collections (e.g. planetary observations or genebank DNA sequences).

3.6.3 SOAP-over-UDP

The results obtained in Section 3.5 show that SOAP-over-UDP provides significantly higher throughput than SOAP-over-HTTP and SOAP-over-TCP. Additionally, it reduces the total message size required to deliver SOAP messages. As a result of this, SOAP-over-UDP is well suited to applications where short SOAP messages are sent frequently and reliability is of less concern. In this type of application, the small amount of data delivered does not justify the overhead of establishing a connection, so making the simple UDP transport appropriate. Secondly, as UDP supports multicasting and broadcasting, it can be used in push-based web services. Some example applications where SOAP-over-UDP is suitable include:

- *Postcode lookup* - where clients provide a location and the web service replies with the corresponding postcode or vice versa.

- *Stock quote push* - where the server broadcast stock prices to a large number of mobile clients. The frequency of such push could vary depending on the movement of stock value.

SOAP-over-UDP also shows certain performance improvement over SOAP-over-HTTP and SOAP-over-TCP for messages with large data payload. To transmit large amount of data using UDP binding, UDP datagram segmentation is used to split a SOAP message into multiple datagrams because a UDP datagram is limited to 65536 bytes. This can be achieved by assigning to all datagrams for the same SOAP message, the same unique Message ID. Each datagram has a Sequence Number to allow easy assembly of the SOAP message on the receiver's end.

3.7 Summary

Advances in wireless technologies and the increasing popularity of Web services have lead to the emergence of mobile Web services. As the main communication protocol for Web services, SOAP plays a very important role in this context. Existing work studied the performance of different SOAP implementations under different environments. However, the effects of using different SOAP bindings have not been extensively studied. The SOAP specification provides an open binding framework that allows SOAP messages to be transported using any underlying protocol. This chapter studied the performance of SOAP-over-HTTP, SOAP-over-TCP and SOAP-over-UDP in wireless networks. While HTTP and TCP have many benefits including the ability to pass through firewalls and reliability, these features may not be needed by all applications. HTTP and TCP perform poorly in wireless environments due to slow connection establishments and packet acknowledgements. Through an extensive set of experiments, it is shown that using the HTTP or TCP binding for SOAP leads to significantly higher overhead than using the UDP binding. Furthermore, the throughput of SOAP-over-UDP is approximately 6 times higher for small messages (around 500 bytes), and 3 times higher for large messages (greater than 1000 bytes), than SOAP-over-HTTP. The performance improvement of UDP scales with the size of the network. These results are attributed to the following:

- UDP does not require a connection to be established to transfer a datagram.
- UDP does not have a strict flow control mechanism as opposed to TCP.
- UDP does not assume packet loss is due to congestion.

SOAP messages that carry a small amount of data can benefit from being sent over wireless networks in UDP datagrams because of low UDP protocol overhead. Depending on the type of each application, a suitable transport protocol for SOAP binding should be selected allowing for efficiency and reliability. To illustrate where various bindings are useful, a number of applications suitable for each binding have been described. UDP binding is well suited to applications that need fast but not necessarily reliable exchange of small data payloads.

Chapter 4

Similarity-based SOAP Multicast Protocol*

SOAP technology is being used in many distributed systems nowadays, however its areas of application are limited by high latency and high protocol overhead. For messaging in environments with high-volume transactions like stock quote and multimedia applications or in mobile and wireless networks, bandwidth-efficient communication is imperative. When there are many transactions requesting similar service operations, using unicast to send SOAP response messages can potentially generate very large amount of traffic. To improve SOAP's performance, SOAP network traffic needs to be reduced. This chapter presents a novel approach to address this issue and to improve overall SOAP performance. A similarity-based SOAP multicast protocol (SMP), which reduces the network load by reducing the total generated traffic size, is proposed. In particular, SMP reuses common templates and payload values among the SOAP messages and only sends one copy of the common part to multiple clients. The results obtained from our experiments indicate that SMP can achieve up to 70% reduction in network traffic compared to traditional SOAP unicast.

4.1 Introduction

SOAP brings extensibility and interoperability to the communication and invocations of services among remote hosts. In contrast to its interoperability advantages, SOAP's major limitation is that its communication produces considerably more network traffic compared to its counterpart technologies such as CORBA and Java-RMI. This issue has drawn great

* Preliminary version of the work presented in this chapter has been published in [Phan et al., 2006b].

interest from many studies to propose techniques enhancing SOAP's performance. Werner et al. [2005] implemented a simple Remote Procedure Call (RPC) server and client to compare the performance of different technologies including MS .Net SOAP, Apache Axis SOAP, CORBA, Java-RMI and RMI-IIOP. Their results showed that both Microsoft's and Apache Axis SOAP implementations typically generate over three times more network traffic than classical approaches like Java-RMI and CORBA.

In the previous chapter, one aspect of SOAP performance drawback in the underlying HTTP transport protocol of SOAP was examined. An alternative SOAP binding with UDP as the transport protocol was proposed to improve the response time and throughput in mobile Web service applications which are run on battery-powered devices and in low bandwidth mobile or wireless networks. SOAP-over-UDP binding performs well in environments with a high number of concurrent clients and small to medium payload messages, but reliability is not guaranteed in these applications. This chapter looks at methods to reduce SOAP traffic for messages with high data payload and without having to compromise for reliability.

4.1.1 Motivation

SOAP messages sent from the same server to multiple clients are generally have some similarity. It is important to emphasize that most SOAP messages have similar byte representations. SOAP messages created by the same implementation generally have the same message structure. Specifically, a SOAP message is surrounded by a large amount of non-domain-related XML data such as name space, encoding specifications and many other XML element names. The number of SOAP implementations is relatively small. Therefore, partial information in a SOAP messages is often known in advance by both receivers and senders. In addition, within business-to-business scenarios, message patterns are very limited and might be known by the server beforehand. Moreover, SOAP responses for the same SOAP request (with the same or different parameter values) will share the same XML document template. Even when the response messages are targeted for different operations of the same service, the response messages may have many similar data type structures.

Bandwidth is expensive in some environments such as mobile and wireless environments and sensors networks. At the current time, there is an increasing demand for the delivery of personalized information such as list of stock quotes, sports scores, weather forecasts and travel information to users, especially to mobile users. If the number of receivers for a service is large (at least several tens of requests) and there is sufficient commonality in their interests, multicast will be an efficient way of delivering information. This is because network resources are used more efficiently by multicasts than broadcasts or replicated unicasts.

In the past, approaches for improving the network bandwidth performance of SOAP Web services focused on the optimization of differential SOAP compression [Werner et al., 2005] and differential SOAP deserialization [Abu-Ghazaleh and Lewis, 2005; Suzumura et al., 2005] techniques. The proposed similarity-based SOAP multicast protocol (SMP) focuses on reducing duplications of common parts between messages.

4.1.2 Statement of the Problem

The main problem of SOAP that has been a focus for a large number of research is to reduce the size of SOAP messages; in turn decreasing the overall traffic caused by sending SOAP messages over a network. This can be done by reducing the size of each single SOAP message individually such as compression or by combining SOAP messages to avoid sending unnecessary duplicate parts of the messages. The proposed technique in this chapter follows the latter approach. A number of issues need to be addressed to achieve the goal of sending aggregated messages to reduce the overall network traffic.

Firstly, the proposed technique must be able to measure the similarity between messages to determine which messages are similar enough to be grouped together in one message. This becomes the problem of measuring the similarity between two and more structured XML documents whose schemas are known in advance. Secondly, a special SMP message structure based on the standard SOAP envelope must be defined to contain the data of multiple recipients in one message.

Thirdly, the SMP solution needs to deal with the processing of aggregated SOAP messages at intermediary SOAP nodes. Currently, when a normal SOAP message is received by a mid-way SOAP node (not the final destination of the SOAP message), the node only parses the SOAP header to carry out any actions required before forwarding the message to the next node. In the SMP solution, special information should be provided in the SMP message header so that SOAP nodes that are not final destinations of individual SOAP messages know how to process the data content before passing it to the following destination. Lastly, the drawback of the SMP approach needs to be realized and compared against with its advantage so that the improvement should outweigh the weakness.

4.1.3 Outline of the Solution

The proposed approach to improving the performance of SOAP messaging is multicasting. The SOAP-based Multicast Protocol (SMP) is introduced, which is based on the *syntactic similarity of SOAP messages*. The nature of SOAP messages, having the same message structure if they are generated by the same service provider, has been utilized by different approaches to improving SOAP performance [Abu-Ghazaleh et al., 2004; Makino et al., 2005; Suzumura et al., 2005; Werner et al., 2004]. However, this feature of Web services has never been used in multicast before. This characteristic is applied to develop SMP.

The multicast technique presented here is built on top of SOAP unicast and does not rely on the low level multicast, so no complex network configuration is required at intermediate routers. In addition, SMP makes use of the commonly available WSDL description of a SOAP Web service when determining the similarity of response messages. For messages that are highly similar, instead of generating messages with duplicated similar parts for different clients, the duplicated parts are reused for multiple clients and are sent only once from the source. In this way, the network traffic can be reduced considerably.

Experiments have been carried out to evaluate our proposed protocol with different network topologies and are detailed in this chapter. Two metrics, response time and traffic load, are used to compare SMP performance with traditional SOAP unicast and multicast. Experiments have shown that a network traffic reduction of over 30% for small networks and up to 70% for large networks and large payload can be obtained using SMP instead of SOAP unicast. This improvement in total network traffic results in an increase in response time by 10% which is an acceptable level.

SOAP's poor performance has been investigated over the past few years. Many studies have proposed approaches for improving the overall performance without compromising interoperability. In particular, a study in [Werner et al., 2004] proposed a compression approach that only the difference between a message and a pre-defined template generated from the WSDL document is sent across the wire. SOAP templates are also utilized by the authors in [Suzumura et al., 2005] to develop a new deserialization method that reuses matching regions from the previously deserialized objects in earlier messages. However, to the best knowledge of the author, no previous study has utilised the similarity between SOAP messages to multicast them as an aggregated message in which the duplicated parts among individual messages appear only once.

In the next section, recent studies on Web services performance improvements utilizing

XML-based SOAP message structure are described. Subsequently, existing multicast and similarity measurement solutions will be studied in Section **??** to provide a basis for describing the contribution of this chapter. SMP will be then comprehensively explained in Section 4.6. Section 4.7 presents the theoretical models of three communication schemes including unicast, multicast and SMP for SOAP as well as the analytical comparisons between these schemes. The performance evaluation and results will be subsequently presented in Section 4.8. The chapter will conclude with a Summary section of discussion and final remarks.

4.2 Related Work

This section discusses various approaches proposed in previous work on improving the performance of Web services based on the similarity of SOAP messages and describes how our method differs from them.

Many studies recognize the fundamental characteristics of Web services — SOAP messages are mostly generated by SOAP engines and there is a high level of similarity between processed messages. By making use of this feature and eliminating redundant processing, researchers have proposed approaches to improve the performance of Web services. Some areas that they have worked on are the XML parser, the serialization and deserialization processes, compression and Web service-Security. Serialization is the process of converting application data into XML messages, while deserialization is the opposite process where XML messages are converted into application objects to be passed to application logic.

Abu-Ghazaleh et al. [2004] proposed a differential serialization technique on the server side in which the entire message is only serialized in the first time. The serialized form of the message is saved after the first dispatch. If subsequent requests ask for the same Web service, it only re-serializes just those elements that have changed and reuses parts of or all of the saved template instead of regenerating it from scratch.

Another research that also takes advantage of the similar structure of SOAP messages is the differential deserialization project done by Suzumura et al. [2005]. They firstly proposed a new XML parser [Takase et al., 2005] which efficiently detects the differential regions between a new XML message and the past messages, and then partially parses only the differential portions. A state machine is used to store state transitions while parsing the XML message. They then extended on that concept to build a deserialization framework in the Web services architecture. Their version of differential deserialization optimizes processing time by recycling in-memory objects on the receiver side of a SOAP communication pair, thereby eliminating the expensive step of creating these objects in memory, achieving substantial performance improvement.

Makino et al.'s WS-Security project focused on byte level similarities in WS-Security messages and implemented a template-based WS-Security processor [Makino et al., 2005]. Templates consist of byte arrays and are stored in an automation. The automation matches the incoming messages against multiple templates in a merged form. If the incoming message matches a template, relevant values such as signatures and encrypted values are extracted from the template. WS-Security processing is subsequently performed by a traditional DOM-based processor and a new template corresponding to the unmatched message is generated. The new template will be then merged into the automation. Their evaluation performance shows that the template-based WS-Security processor performs 60% better than the stream-based processor at the server side and 44% faster than the Document Object Model (DOM)-based processor at the client side. The high performance of the proposed template-based processor is assisted by the elimination of costly XML parsing in either the DOM or SAX (Simple API for XML) fashion.

Another important contribution to reducing network traffic caused by SOAP is from Werner et al. [2005]. The authors proposed a differential SOAP compression technique that is based on the commonly available Web service's WSDL description document. In their method, the WSDL document of a Web service is used to generate SOAP messages' skeletons which contain only the markup, for all defined service operations. On the server-side, the skeletons are used to create SOAP difference documents which include only the differences between a SOAP message and its corresponding skeleton files. These difference documents are very small in size and are sent over the wire. On the client-side, the skeleton files are used with the difference files to reconstruct the original SOAP messages. They use two tools *diffxml* [Mouat, 2002] and *XUpdate* [Andreas Laux, 2000] for creating difference documents. Their experiments show that their differential encoding technique is advantageous only when the the SOAP message' markup is predictable. When the payload amount increases, the structure of the SOAP messages cannot be predicted accurately, leading to minor improvement in the resulting message size of a compressed difference document compared to a compressed normal SOAP message.

4.3 Background

In this section, background on an existing multicast protocol that has similar approach as SMP and common XML similariy measurement methods is presented.

4.3.1 Explicit Multicast Protocols

Xcast [Boivie et al., 2000] has been proposed to solve the problems with using IP multicast for small multicast groups. Xcast multicast scheme aims to support a very large number of small multicast groups by explicitly including the list of destinations in packets, instead of using a multicast address. The source stores a destination list in the Xcast header and then sends the packet to a router. Each downstream router parses the header, partitions the destinations based on each destination's next hop, and forwards a packet with an appropriate Xcast header along each of the next hops (see Figure 4.1). Each final router removes the

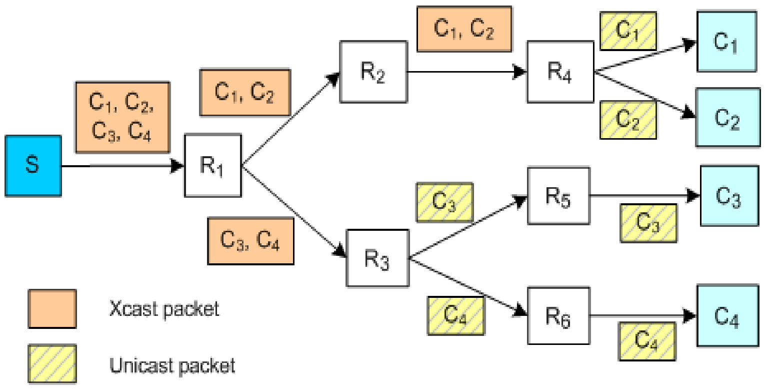


Figure 4.1: Xcast Routing Mechanism

Xcast encoding and forwards the data to its destination as a standard unicast packet. There is no need to maintain multicast states in the network because Xcast carries the state in each data packet and processes it in real time along the forwarding path.

Xcast performs well with small size multicast groups, but if the number of members in a group increases, the destination addresses cannot be sufficiently encoded in an Xcast packet header due to size limitation. Xcast+ [Shin et al., 2001] has been proposed to support a large number of medium size multicast groups. In Xcast+, every source or destination is associated with a Designated Router (DR). Instead of encoding the set of group members in the Xcast packet, Xcast+ encodes the set of their DRs. There are other variations and improved versions of Xcast, for example GXcast [Boudani et al., 2004] and SEM [Boudani and Cousin, 2003].

The proposed protocol, SMP, differs from others in that it utilizes the similar structure of SOAP messages to multicast messages in a more efficient way. Like Xcast, SMP uses unicast routing and encodes a list of clients inside the message, instead of using a multicast address. However, Xcast can only multicast identical messages, while SMP can also send slightly different ones together, as explained in Section 4.6.3.

4.3.2 Similarity Measurements

Two popular techniques for measuring document similarities are presented here.

Levenshtein's Edit Distance

The *edit distance* formulated by Levenshtein [1966] is a well-established method for weighting the difference between two strings. It measures the minimum number of token insertions, deletions, and substitutions required to transform one string into another using a dynamic programming algorithm. For example, the edit distance, ed, between the two strings "$3-1$" and "$3-0$" equals 1, $ed(\text{"}3-1\text{"}, \text{"}3-0\text{"}) = 1$, because one substitution is required to change the string "$3-1$" into "$3-0$".

Syntactic Similarity Measures

Maedche and Staab [2001] proposed a syntactic similarity measure for strings which is based on Levenshtein's edit distance. It is defined as follows:

Definition 2 *Syntactic similarity*: Given two strings s_i and s_j, their syntactic similarity is defined as: $SM(s_i, s_j) = max(0, \frac{min(|s_i|,|s_j|)-ed(s_i-s_j)}{min(|s_i|,|s_j|)})$, where

- SM is the syntactic similarity measure, which has a value between 0 and 1;
- $|s_i|$ and $|s_j|$ are the lengths of two strings s_i and s_j respectively; and
- ed is the edit distance.

This formula has been widely used in many ontological studies. This measurement considers the number of changes that are required to change one string into the other and weighs the number of these changes against the length of the shortest string. In the previous example, $SM(\text{"}3-1\text{"}, \text{"}3-0\text{"}) = \frac{2}{3})$ is obtained.

4.4 Similarity Measurement Model for Clustering SOAP Messages

The goal of clustering is to group SOAP messages based on their syntactic similarities. By identifying the similarities in the message structures of the SOAP response messages, the source can determine which messages can be sent together using SMP multicast technique. The ability to measure the similarity between messages is paramount to determining if they have enough parts in common to justify the cost of extra processing time required at the source and routers. This section describes the similarity measurement model used in SMP.

The similarity between two messages is measured based on the XML schema of each SOAP message. The XML schema of a message is the WSDL document for a service, provided by a service provider to service consumers. The WSDL document defines a list of operations and their signatures. It also defines the complex data types that are used in operations. Response messages to any request for service operations are formed from these complex data type definitions.

A stock quote application will be used as examples throughout this chapter to explain the proposed similarity measurement and SMP routing methods. The stock quote application involves a large number of transactions requesting the latest stock prices. The responses for these requests would likely have similar message structures. Therefore, the chosen application satisfies our requirements for SMP's applicable services. The real-time stock quote Web service receives requests to retrieve stock prices, stock statistics over a day and general company information. The service provides a list of operations such as getStockQuote(), getQuoteAndStatistic(), getCompanyInfo(), getFullQuote() and getMarketInfo(). The inputs for these requests can be a string of more than one stock symbol.

Two SOAP messages, given in Figures 4.2 and 4.3 are used to illustrate how the similarity between two XML documents is measured. These are two response messages to operations *getQuote()* and *getQuoteAndStatistic()* respectively. For simplicity, the SOAP messages in this example are element-based and do not contain attributes. In reality, attributes can be considered by treating them as sub-elements.

4.4.1 Foundation Definitions

Basic concepts in a SOAP message document are presented here as the foundation for the proposed similarity measurement method. Since a SOAP message is based on XML, it is a hierarchical document with nested tags. As such, a SOAP message is best modeled as a rooted ordered tree.

```
<StockQuoteResponse>
<ArrayOfStockQuote>
 <StockQuote>
  <Symbol>NAB</Symbol>
  <QuoteInfo>
    <Price>28.56</Price>
    <LastUpdated>24/01/2007 10:15am</LastUpdated>
  </QuoteInfo>
 </StockQuote>
 <StockQuote>
  <Symbol>BHP</Symbol>
  <QuoteInfo>
    <Price>24.52</Price>
    <LastUpdated>24/01/2007 10:45am</LastUpdated>
  </QuoteInfo>
</StockQuote>
</ArrayOfStockQuote>
</StockQuoteResponse>
```

Figure 4.2: $Soap_1$ message: A simple stock quote response message to the getStock-Quote("NAB, BHP") request.

Definition 3 A SOAP message tree T is a rooted ordered tree, such that each node ϵ_T is a pair $\epsilon_T = (\eta, \upsilon)$ where η is the node name and υ is the node value. Value υ can be of primitive type (Boolean, Float, Double andDate) or complex type (which may include one or more primitive types).

Elements of a SOAP message tree are defined as:

Definition 4 [*Element*]: An element (Ele) represents a simple or complex node in a SOAP message. Each leaf element, also called simple element, represents one feature of a complex object. A non-leaf element is a set of features which describe one facet of the complex object.

In our example of $Soap_1$ and $Soap_2$ messages, some complex elements are "StockQuote", "QuoteInfo", "QuoteAndStatistic" and "Statistic". Other elements such as "Symbol", "Price", "LastUpdated" and "Change" are simple elements.

An instance of an element is defined as below.

Definition 5 [*Element instance*]: An XML element instance (noted as *inst*) instantiates an element of the schema.

For example, in the $Soap_1$ message, Figure 4.2, there are two instances of the "StockQuote" element: one for 'NAB' symbol and one for 'BHP'

```
<QuoteAndStatisticResponse>
<ArrayOfQuoteAndStatistic>
 <QuoteAndStatistic>
  <Symbol>BHP</Symbol>
  <QuoteInfo>
   <Price>24.52</Price>
   <LastUpdated>24/01/2007 10:45am</LastUpdated>
  </QuoteInfo>
  <Statistic>
   <Change>+0.50</Change>
   <OpenPrice>24.02</OpenPrice>
  </Statistic>
 </QuoteAndStatistic>
 <QuoteAndStatistic>
  <Symbol>NAB</Symbol>
  <QuoteInfo>
    <Price>28.56</Price>
    <LastUpdated>24/01/2007 10:15am</LastUpdated>
  <Statistic>
    <Change>-0.10</Change>
    <OpenPrice>28.66</OpenPrice>
  </Statistic>
 </QuoteAndStatistic>
 </ArrayOfQuoteAndStatistic>
 </QuoteAndStatisticResponse>
```

Figure 4.3: $Soap_2$ message: A SOAP response message to the getQuoteAndStatistic("BHP, NAB") request to get both stock quotes and their market statistics.

An element is called a multi-instance element if there are more than one instance of the element, and is called a single-instance element if there is only one instance of the element in an XML document. In Figure 4.2, "ArrayOfStockQuote" and "ArrayOfQuoteAndStatistic" are single-instance elements while "StockQuote", "QuoteAndStatistic", "Symbol" and "Price" are multi-instance elements. An instance of the "Price" element in the $Soap_1$ message is "<*Price*>28.56<*Price*>" (for symbol 'NAB').

Definition 6 [*Value*]: A value is the value (V) of a leaf element instance.

The value of the first instance of *Symbol* in the $Soap_1$ message in Figure 4.2 is 'NAB'.

Next, the computation for the instance similarity of a simple element between two instances of the element is presented.

Definition 7 [*Instance similarity of simple element*]: Simple element instance similarity, noted as Sim^E_{inst}, of two instances of the same simple element, namely E, is the similarity of leaf instances' values.

For different data types of a value, different methods can be used to calculate instance similarity of simple element. There are a number of built-in data types supported in XML schema [W3C, 2007b] such as *String, Boolean, Float, Double* and *Date.* For each data type, different methods should be used to calculate the instance similarity of simple element. The methods presented by Ma and Chbeir [2005] are adopted to define the similarity in value of two nodes that have the same simple data type because they are widely used by other authors.

The following definition gives the similarity between two numeric values. A data type is said to be numeric if its values are conceptually quantities [W3C, 2007b].

Definition 8 Given two numeric values NV_1 and NV_2, the similarity between them is defined as below:

$sim(NV_1, NV_2) = 1 - \frac{|NV_1 - NV_2|}{|NV_1| + |NV_2|}$, where $|A|$ is the absolute value of numeric value A.

The similarity between two boolean values can be measured as follows:

Definition 9 Given two boolean values BV_1 and BV_2, their similarity is defined as:

$$sim(BV_1, BV_2) = NOT(XOR(BV_1, BV_2))$$

The similarity between two strings is computed using the method proposed in [Maedche and Staab, 2001], which was discussed in the Background section (see Definition 2).

The syntactic similarity is the only measurement of interest here because the objective is to find common contents among SOAP messages to apply the SMP routing algorithm so that the total size of traffic can be reduced. By applying the method given in Definition 2 to calculate the similarity between two strings to Definition 7, similarities between different "*Symbol*" instances in the $Soap_1$ and $Soap_2$, given in Figures 4.2 and 4.3, can be computed as follows:

$$\begin{aligned}
Sim_{inst}^{\text{“}Symbol\text{”}}(NAB^{Soap_1}, BHP^{Soap_2}) &= max(0, \tfrac{min(|\text{‘}NAB'|,|\text{‘}BHP'|)\ -\ ed(\text{‘}NAB' - \text{‘}BHP')}{min(|\text{‘}NAB'|,|\text{‘}BHP'|)}) \\
&= 0 \\
Sim_{inst}^{\text{“}Symbol\text{”}}(NAB^{Soap_1}, NAB^{Soap_2}) &= max(0, \tfrac{min(|\text{‘}NAB'|,|\text{‘}NAB'|)\ -\ ed(\text{‘}NAB' - \text{‘}NAB')}{min(|\text{‘}NAB'|,|\text{‘}NAB'|)}) \\
&= 1 \\
Sim_{inst}^{\text{“}Symbol\text{”}}(BHP^{Soap_1}, BHP^{Soap_2}) &= 1 \\
Sim_{inst}^{\text{“}Symbol\text{”}}(BHP^{Soap_1}, NAB^{Soap_2}) &= 0
\end{aligned}$$

The element similarity of a particular element in two SOAP messages is defined as the average of all instance similarities of all instances of E contained in the two messages.

Definition 10 The element similarity of an element, namely E, in two SOAP messages $Soap_1$ and $Soap_2$ is defined as:

$$Sim_{ele}^{E}(Soap_1, Soap_2) = \frac{\sum_{i=1}^{N_{max}} Sim_{inst}^{E}(E_i^{Soap_1}, E^{Soap_2})}{N_{max}}, \text{ where}$$

- $|E^{Soap_1}|and|E^{Soap_2}|$ are the numbers of element E instances in $Soap_1$ and $Soap_2$ respectively;
- $N_{max} = Max(|E^{Soap_1}|, |E^{Soap_2}|)$; and
- $Sim_{inst}^{E}(E_i^{Soap_1}, E^{Soap_2}) = Max(Sim_{inst}^{E}(E_i^{Soap_1}, E_j^{Soap_2})), \forall E_j^{Soap_2} \in Soap_2$, which is the similarity of instance $E_i^{Soap_1}$ to all instances of E in $Soap_2$.

For single-instance elements, the element similarity is equal to its instance similarity. This definition applies to both simple and complex element types. In the example, the element similarity for the element "Symbol" between the $Soap_1$ and $Soap_2$ example messages can be computed as follows:

$$Sim_{ele}^{\text{``Symbol''}}(Soap_1, Soap_2) = \frac{\sum_{i=1}^{2} Sim_{inst}^{\text{``Symbol''}}(\text{``Symbol''}_i^{Soap_1}, \text{``Symbol''}^{Soap_2})}{2}$$
$$= \frac{1}{2}(Sim_{inst}^{\text{``Symbol''}}(NAB^{Soap_1}, \text{``Symbol''}^{Soap_2}) + Sim_{inst}^{\text{``Symbol''}}(BHP^{Soap_1}, \text{``Symbol''}^{Soap_2}))$$
$$= \frac{1}{2}(Max(Sim_{inst}^{\text{``Symbol''}}(NAB^{Soap_1}, BHP^{Soap_2}), Sim_{inst}^{\text{``Symbol''}}(NAB^{Soap_1}, \text{`}NAB^{Soap_2}))$$
$$+ Max(Sim_{inst}^{\text{``Symbol''}}(BHP^{Soap_1}, BHP^{Soap_2}), Sim_{inst}^{\text{``Symbol''}}(BHP^{Soap_1}, \text{`}NAB^{Soap_2})))$$
$$= \frac{(1+1)}{2} = 1$$

The following definition gives a formula for computing the instance similarity between two instances of a complex element.

Definition 11 The instance similarity between two instances P_1 and P_2 of a complex element P is calculated based on the element similarities of all their sub-elements E_i, as defined below:

$$Sim_{inst}^{P}(P_1, P_2) = \frac{\sum_{i=1}^{N} Sim_{ele}^{E_i}(P_1, P_2)}{N}, \text{ where}$$

- N is the number of sub-elements in the complex element P; and
- $Sim_{ele}^{E_i}(P_1, P_2)$ is the element similarity of element E_i in instance P_1 to E_i in instance P_2.

For example, the instance similarity of the complex element "*QuoteInfo*" between the first instances of "*QuoteInfo*" in the $Soap_1$ and $Soap_2$ messages can be computed as:

$$\begin{aligned}
& Sim_{inst}^{\text{``}QuoteInfo\text{''}}(\text{QuoteInfo}^{\text{NAB}}, \text{QuoteInfo}^{\text{BHP}}) \\
= \; & \frac{1}{2}(Sim_{ele}^{\text{``}Price\text{''}}(\text{QuoteInfo}^{\text{NAB}}, \text{QuoteInfo}^{\text{BHP}}) + Sim_{ele}^{\text{``}LastUpdated\text{''}}(\text{QuoteInfo}^{\text{NAB}}, \text{QuoteInfo}^{\text{BHP}})) \\
= \; & \frac{1}{2}(0.92 + 1) = 0.96.
\end{aligned}$$

4.4.2 Similarity between Two SOAP Messages

The method used to compute the overall similarity between two full SOAP messages is presented here. Most SOAP response messages, which are generated by the same service provider, will have a common message structure (or template). In particular, they will have similar information in the header and the same structure in the body, if they have been generated for the same operation. Based on this observation, the similarity between two SOAP messages $Soap_1$ and $Soap_2$, labeled as $sim(Soap_1, Soap_2)$, is defined as the product of the similarity between the two message templates and the similarity between the data values of the messages.

Definition 12 The similarity between two SOAP messages $Soap_1$ and $Soap_2$ is defined as:

$$sim(Soap_1, Soap_2) = sim_{temp}(Soap_1, Soap_2) \times sim_{val}(Soap_1, Soap_2), \text{ where}$$

- $sim(Soap_1, Soap_2) \in$ [0,1];
- $sim_{temp}(Soap_1, Soap_2)$ is the similarity between $Soap_1$'s and $Soap_2$'s templates; and
- $sim_{val}(Soap_1, Soap_2)$ is the similarity between $Soap_1's$ and $Soap_2's$ data values.

SOAP messages are based on XML templates which are formed by a set of elements. The similarity between two SOAP message templates is measured based on the common XML nodes between the two XML trees of the SOAP messages.

Definition 13 The similarity between $Soap_1$'s and $Soap_2$'s templates is defined as:

$$sim_{temp}(Soap_1, Soap_2) = \frac{|N(Soap_1) \bigcap N(Soap_2)|}{|N(Soap_1) \bigcup N(Soap_2)|}, \text{ where}$$

- N(*xml_tree*) is the set of *distinctive* xml nodes in the *xml_tree* template; and
- $|S|$ is the cardinality of set S.

It is noteworthy to emphasize that only distinctive nodes (excluding closing tags) of a SOAP template are included in the above computation. In relation to the StockQuote example, the set of common nodes between $Soap_1$ and $Soap_2$ templates is:

$$N(Soap_1) \cap N(Soap_2) = \{Symbol, QuoteInfo, Price, LastUpdated\} \tag{4.1}$$

The set of all distinctive nodes in both $Soap_1$'s and $Soap_2$'s templates is:

$$\begin{aligned} N(Soap_1) \cup N(Soap_2) = \{&StockQuoteResponse, ArrayOfStockQuote, StockQuote, Symbol, \\ &QuoteInfo, Price, LastUpdated, QuoteAndStatisticResponse, \\ &ArrayOfQuoteAndStatistic, QuoteAndStatistic, Statistic, \\ &Change, OpenPrice\} \end{aligned}$$

Therefore, the similarity in *template* between the two example messages is:

$$Sim_{temp}(Soap_1, Soap_2) = \frac{|N(Soap_1) \bigcap N(Soap_2)|}{|N(Soap_1) \bigcup N(Soap_2)|} = \frac{4}{13} \tag{4.2}$$

The following definition shows how the data values in two SOAP documents are compared.

Definition 14 The similarity between $Soap_1$'s and $Soap_2$'s data values is defined as:

$$sim_{val}(Soap_1, Soap_2) = \frac{\sum_{i=1}^{|M|} sim_{ele}^{E_i}(Soap_1, Soap_2)}{|M|}, \text{ where}$$

- M is the set of common distinctive nodes but excluding sub-element nodes between $Soap_1$ and $Soap_2$ messages;
- E_i is an element in M;
- $|M|$ is the cardinality of set M; and
- $sim_{ele}^{E_i}(Soap_1, Soap_2)$ is the element similarity between two values of the common node E_i in both $Soap_1, Soap_2$.

It is important to note that only the values of the non-subelements nodes in the set $|N(Soap_1) \cap N(Soap_2)|$ are compared. Nodes that are not common between two messages are ignored in the value similarity computation because it is unlikely that two nodes of different data types have similar values. Sub-element nodes of a common node are not included in the computation either because their similarity is already considered during the similarity computation of their common parent node. Back to our example, among elements in the common node set between the $Soap_1$ and $Soap_2$ messages, the set M will include only "Symbol" and "QuoteInfo". The similarity in value between the two SOAP messages $Soap_1$ and $Soap_2$ in our example can be computed as follows:

$$\begin{aligned} Sim_{val}(Soap_1, Soap_2) &= \frac{1}{2}(Sim_{ele}^{\text{``}Symbol\text{''}}(Soap_1, Soap_2) + Sim_{ele}^{\text{``}QuoteInfo\text{''}}(Soap_1, Soap_2)) \\ &= \frac{1}{2}(1+1) = 1 \qquad (4.3) \end{aligned}$$

Combining Equations 4.2 and 4.3 into Definition 12, the overall similarity between the two $Soap_1$ and $Soap_2$ messages is given by:

$$Sim(Soap_1, Soap_2) = \frac{4}{13} \times 1 = 0.3$$

Two SOAP messages are considered syntactically similar if their similarity degree is equal to or greater than a *similarity threshold*, denoted by ρ, which is dependent on the domain of each application.

Another example is given here to further illustrate the proposed similarity measurement model. $Soap_3$ (see Figure 4.4) is another sample SOAP response message to a *getStockQuote("NAB,WIL,BHP")* request. Since $Soap_1$ and $Soap_3$ have the same message structure, their similarity in template is 1: $Sim_{temp}(Soap_1, Soap_3) = 1$.

For brevity, in the following equations *"SQR"* is used for ' '*StockQuoteResponse*", *"AOSQ"* for *"ArrayOfStockQuote"* and *"SQ"* for *"StockQuote"*.

$$\begin{aligned} Sim_{val}(Soap_1, Soap_3) &= Sim_{ele}^{\text{``}SQR\text{''}}(Soap_1, Soap_3) = Sim_{inst}^{\text{``}AOSQ\text{''}}(Soap_1, Soap_3) \\ &= Sim_{ele}^{\text{``}SQ\text{''}}(Soap_1, Soap_3) \\ &= \frac{1}{3}(Sim_{inst}^{\text{``}SQ\text{''}}(SQ_{NAB}^{Soap_3}, SQ^{Soap_1}) + Sim_{inst}^{\text{``}SQ\text{''}}(SQ_{WIL}^{Soap_3}, SQ^{Soap_1}) \\ &+ \quad Sim_{inst}^{\text{``}SQ\text{''}}(SQ_{BHP}^{Soap_3}, SQ^{Soap_1})) \\ &= \frac{1}{3}(1 + Sim_{inst}^{\text{``}SQ\text{''}}(SQ_{WIL}^{Soap_3}, SQ^{Soap_1}) + 1) \qquad (4.4) \end{aligned}$$

The instance similarity between the *"StockQuote"* instance that contains symbol *WIL* in the $Soap_3$ message and all instances of *"StockQuote"* in the $Soap_1$ message is computed as

```
<StockQuoteResponse>
<ArrayOfStockQuote>
 <StockQuote>
  <Symbol>NAB</Symbol>
  <QuoteInfo>
    <Price>28.56</Price>
    <LastUpdated>24/01/2007 10:15am</LastUpdated>
  </QuoteInfo>
 </StockQuote>
 <StockQuote>
  <Symbol>WIL</Symbol>
  <QuoteInfo>
    <Price>1.29</Price>
    <LastUpdated>24/01/2007 10:45am</LastUpdated>
  </QuoteInfo>
 </StockQuote>
  <StockQuote>
  <Symbol>BHP</Symbol>
  <QuoteInfo>
    <Price>24.52</Price>
    <LastUpdated>24/01/2007 10:45am</LastUpdated>
  </QuoteInfo>
</StockQuote>
</ArrayOfStockQuote>
</StockQuoteResponse>
```

Figure 4.4: Soap$_3$ message: A simple stock quote response message to the getStock-Quote("NAB, WILL, BHP") request

.

follows:

$$\begin{aligned} & Sim_{inst}^{\text{“}SQ\text{”}}(SQ_{WIL}^{Soap_3}, SQ^{Soap_1}) \\ = \; & Max(Sim_{inst}^{\text{“}SQ\text{”}}(SQ_{WIL}^{Soap_3}, SQ_{NAB}^{Soap_1}), Sim_{inst}^{\text{“}SQ\text{”}}(SQ_{WIL}^{Soap_3}, SQ_{BHP}^{Soap_1})) \qquad (4.5) \end{aligned}$$

First, the *"StockQuote"* instance that contains symbol *WIL* in the $Soap_3$ message is compared with the *"StockQuote"* instance that contains symbol *NAB* in the $Soap_1$ message.

$$\begin{aligned}
Sim_{inst}^{\text{"}SQ\text{"}}(SQ_{WIL}^{Soap_3}, SQ_{NAB}^{Soap_1}) &= \frac{1}{2}(Sim_{ele}^{\text{"}Symbol\text{"}}(SQ_{WIL}^{Soap_3}, SQ_{NAB}^{Soap_1}) \\
&+ \quad Sim_{ele}^{\text{"}QuoteInfo\text{"}}(SQ_{WIL}^{Soap_3}, SQ_{NAB}^{Soap_1})) \\
&= \frac{1}{2}(0 + Sim_{inst}^{QuoteInfo}(SQ_{WIL}^{Soap_3}, SQ_{NAB}^{Soap_1})) \\
&= \frac{1}{2}(0 + \frac{1}{2}(Sim_{inst}^{Price}(SQ_{WIL}^{Soap_3}, SQ_{NAB}^{Soap_1}) \\
&+ \quad Sim_{inst}^{LastUpdated}(SQ_{WIL}^{Soap_3}, SQ_{NAB}^{Soap_1}))) \\
&= \frac{1}{4}(0.08 + 0.94) = 0.25 \qquad (4.6)
\end{aligned}$$

Next, the instance similarity between $SQ_{WIL}^{Soap_3}$ and $SQ_{BHP}^{Soap_1}$ is calculated.

$$\begin{aligned}
Sim_{inst}^{\text{"}SQ\text{"}}(SQ_{WIL}^{Soap_3}, SQ_{BHP}^{Soap_1}) &= \frac{1}{2}(Sim_{ele}^{\text{"}Symbol\text{"}}(SQ_{WIL}^{Soap_3}, SQ_{BHP}^{Soap_1}) \\
&+ \quad Sim_{ele}^{\text{"}QuoteInfo\text{"}}(SQ_{WIL}^{Soap_3}, SQ_{BHP}^{Soap_1})) \\
&= \frac{1}{2}(0 + Sim_{inst}^{QuoteInfo}(SQ_{WIL}^{Soap_3}, SQ_{BHP}^{Soap_1})) \\
&= \frac{1}{2}(0 + \frac{1}{2}(Sim_{inst}^{Price}(SQ_{WIL}^{Soap_3}, SQ_{BHP}^{Soap_1}) \\
&+ \quad Sim_{inst}^{LastUpdated}(SQ_{WIL}^{Soap_3}, SQ_{NAB}^{Soap_1}))) \\
&= \frac{1}{4}(0.1 + 1) = 0.275 \qquad (4.7)
\end{aligned}$$

From Equations 4.7 and 4.6, it can be drawn that $Sim_{inst}^{\text{"}SQ\text{"}}(SQ_{WIL}^{Soap_3}, SQ^{Soap_1}) = 0.275$. Placing this value into Equation 4.4, a value for $Sim_{val}(Soap_1, Soap_3)$ of approximately 0.76 is obtained, which is also the overall similarity between the $Soap_1$ and $Soap_3$ messages.

4.5 SOAP Message Tree Indexing

An indexing method is proposed in this section to index SOAP element tags in messages, thus reducing message size. SOAP messages are generated based on a set of data types defined in a WSDL file, which is an XML-based document describing Web services and how they can be accessed. Both the server and clients generally have access to the WSDL document before the communication between them is initiated. Clients need to have the WSDL description for the service, so that they can send correctly formatted requests to the server and can de-serialise the server's responses to get the data they want to retrieve. In this thesis, it is assumed that clients have a copy of the WSDL document of the service they request. The WSDL specification for the Stock Quote service used in this chapter's examples is given in Appendix A.

A WSDL document specifies names, structures, and types for all the data types that the SOAP service exposes. A SOAP response message typically consists of some of those data types and their values. Let us consider the $Soap_1$ message consisting of a complex data type <*StockQuote*> (which includes another complex data type $<QuoteInfo>$) and a simple data type <*Symbol*>. The <*StockQuote*> element is a multiple instance element, which means that there may be multiple instances of <*StockQuote*> in SOAP messages. Since the clients have access to the XML structure definitions of all SOAP elements from the WSDL description, it is redundant to send the static data type templates over the network. Instead of sending full SOAP messages including templates and data, the indexed forms of the SOAP messages, which are more compact than standard SOAP messages, are sent to the clients. The indexed form of a SOAP message contains the identifiers (ID) of each element in the message template as well as the data values of the elements.

As given in Definition 3, a SOAP message is an ordered XML tree. In the context of this thesis, the term "SOAP messages" refers to SOAP response messages to clients' requests as response messages are typically much larger in size than request messages and generate high network traffic. The main objective of the algorithms presented in this thesis is to reduce the traffic generated by SOAP response messages.

Each data type definition specified in a WSDL document is indexed with a unique identifier (ID). The occurrences of a data type in SOAP messages are represented by its ID, which is much smaller than its full name. Nodes in a SOAP XML tree are labeled using a Dewey numbering system [Dewey, 2004] that indicates the location of the nodes in the tree. Nodes are listed in the order that they are visited by the Breadth-First-Search (BFS) algorithm[1] [Levitin, 2007]. Dewey's notations are used to label XML documents in many research areas of pattern matching, searching and database [Lu et al., 2005; Zhang et al., 2005]. Specifically, each node in a SOAP message tree is encoded in the format given in Definition 15 to become an element in the equivalent indexed SOAP message.

Definition 15 A node in a SOAP document is indexed in the following format:

$$<v\ pos = \text{`}a\ position'\ eRef = \text{`}an\ ID'\ val = \text{`}a\ value'>, \text{ where}$$

[1] BFS is a graph search algorithm that begins at the root node and traverses all the nearest nodes. Then for each of these neighboring nodes, it explores their neighboring nodes that have not been visited, and so on until the goal is found.

- The *pos* attribute is the position of the node in the SOAP message tree;
- The *eRef* attribute is the unique identifier of the data type at the node as defined in the WSDL document; and
- The *val* attribute is the value at the node. Depending on the node type, the data stored in *val* will be different. If the node is a:
  - Normal node: *val* stores the data value of the node;
  - Array node: *val* stores the number of elements in the array;
  - Complex node: *val* is empty.

Figures 4.5 and 4.6 show the tree representations of $Soap_1$ and $Soap_3$ messages with node labels. The labels are superscripted in these figures.

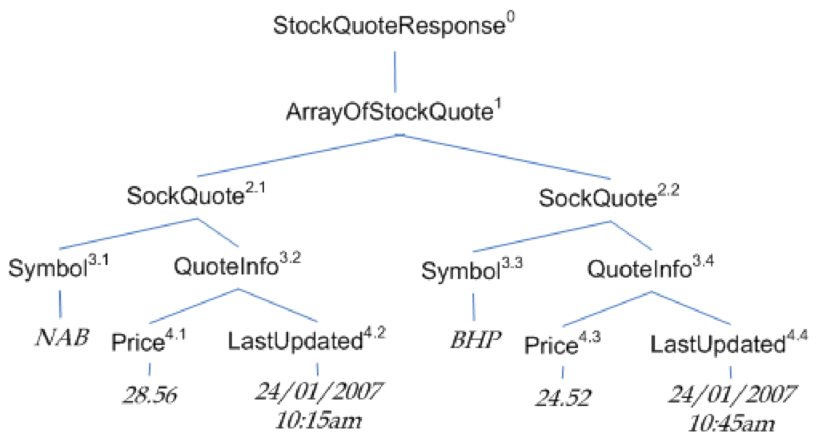


Figure 4.5: $Soap_1$ XML tree, a response to getStockQuote("NAB, BHP") request, with node labels.

The full indexed version[2] of $Soap_1$ is given in Figure 4.7. The <*ArrayOfStockQuote*> node is encoded as `<v pos='1' eRef='AOSQ' val='2'>` because the node is at position 1 in the tree, the *ArrayOfStockQuote* data type is indexed as '*AOSQ*' in the WSDL document, and there are 2 elements of *StockQuote* in its array. The first <*StockQuote*> node instance in $Soap_1$ is encoded as `<v pos='2.1' eRef='SQ' val=''>` (node <*StockQuote*> is at position 2.1, its data type reference in the WSDL is 'SQ', and its value is empty because it is a complex

[2] For brevity, examples of SOAP messages described in this chapter display only the body contents of the messages, the headers are omitted.

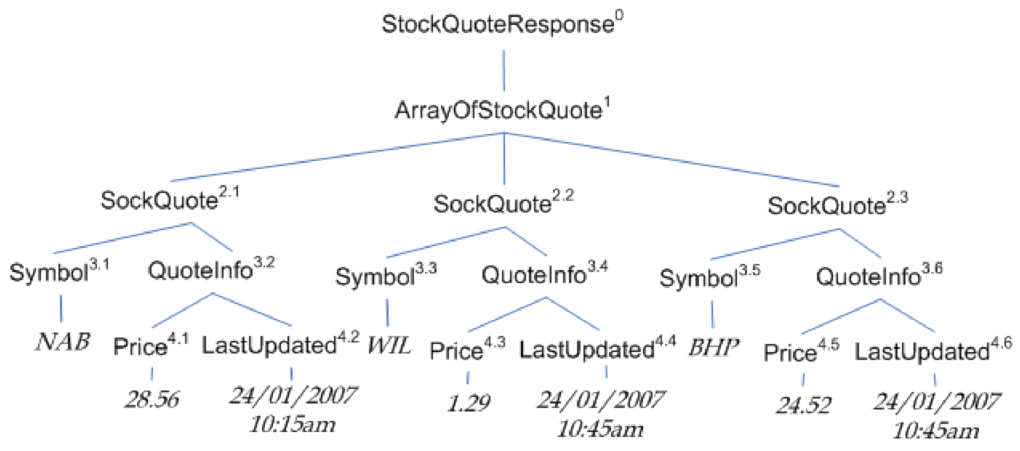


Figure 4.6: Soap$_3$ XML tree, a response to getStockQuote("NAB, WIL, BHP") request, with node labels.

node.) By using this tree encoding technique, the size of a SOAP message can be significantly reduced and so is the overall network traffic.

4.6 Similarity-based SOAP Multicast Protocol (SMP) Solution

The concepts and operations of SMP are explained in detail in this section. The proposed similarity-based SOAP multicast protocol is designed to deal with SOAP performance issues by exploiting the similar structure of SOAP messages. The goal is to reduce the total traffic generated over a network when sending SOAP responses from servers to clients. SMP allows similar SOAP messages that share some parts of the SOAP template to be sent in one customised SMP message instead of being sent as multiple copies.

4.6.1 Design Goals and Assumptions

The objective when designing the SMP protocol is to base it on existing SOAP protocol standards so that it can integrate seamlessly with the current Web services framework. SMP should require minimal administration efforts at intermediary routers in order to enable fast deployment and easy maintenance. It is important that SMP scales well with a large number of clients. The proposed approach is to send the common message structure and data values of multiple messages and their distinctive parts in a single message instead of sending multiple messages containing duplicated copies of common parts. An important goal is to have SMP support the aggregation of multiple messages for a large number of clients.

```
<v pos='0' eRef='SQR'>
<v pos='1' eRef='AOSQ' val='2'>
<!--First StockQuote instance -->
<v pos='2.1' eRef='SQ' val=''>
<v pos='3.1' eRef='Sbl' val='NAB'>
<v pos='3.2' eRef='QI'>
<v pos='4.1' eRef='P' val='28.56'>
<v pos='4.2' eRef='LU' val='24/01/2007 10:15am'>
<!--Second StockQuote instance -->
<v pos='2.2' eRef='SQ'>
<v pos='3.3' eRef='Sbl' val='BHP'>
<v pos='3.4' eRef='QI'>
<v pos='4.3' eRef='P' val='24.52'>
<v pos='4.4' eRef='LU' val='24/01/2007 10:45am'>
```

Figure 4.7: Indexed version of the $Soap_1$ message

4.6.2 SMP Message Structure and Generation

This section explains the structure of an SMP message and how it is generated.

SMP Message Structure

Clients' addresses are represented as strings and stored in the SMP header, which is encapsulated inside the SOAP message body, as shown in Figure 4.8. The SMP body is also embedded inside the SOAP message body. There are two sections in the SMP body: (1) the <*Common*> section containing common values and structures of all messages addressed to clients encoded in the SMP header; (2) the <*Distinctive*> section containing individual different parts for each response message. The outermost envelope is referred to as an SMP message. The destination of an SMP message, which is specified in the SOAP header, is the next router in a network when the message is forwarded to all clients given in its SMP header.

SOAP message Aggregation

This section describes how two SOAP messages can be merged to become an SMP message. The generation of an SMP message is explained through an example based on the network configuration shown in Figure 4.11. In this example, clients C_1 and C_5 make *getStockQuote("NAB,BHP")* request to the server to retrieve the quotes for NAB and BHP stocks. Clients C_2 and C_4 request the quotes for three stocks: NAB, WIL and BHP via the *getStockQuote("NAB,WIL,BHP")* request. Client C_3 calls for *getStockQuote("NAB")*

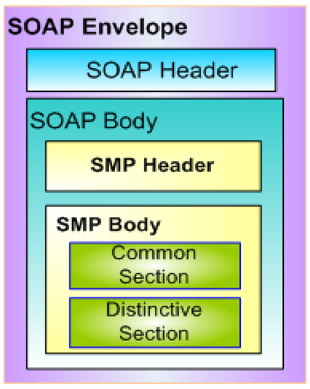


Figure 4.8: SMP Envelope embedded inside a SOAP envelope

request.

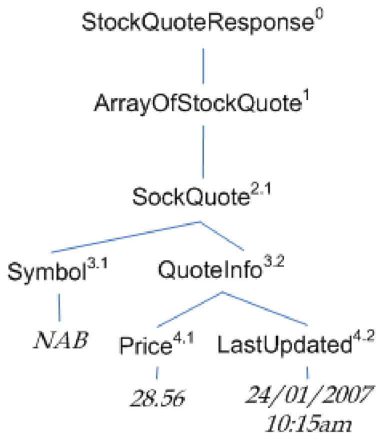


Figure 4.9: $Soap_4$ XML tree, a response to the getStockQuote("NAB") request, with node labels.

Figures 4.5, 4.6 and 4.9 illustrate the hierarchical SOAP trees for messages responding to the *getStockQuote("NAB,BHP")* and *getStockQuote("NAB, WIL,BHP")* and *getStockQuote("NAB")* requests respectively. It is assumed here that the request messages arrive at the server in the order of requests from clients C_1, C_2, C_3, C_4, and C_5. Upon receiving similar requests from clients C_1 and C_2, and determining that the similarity of the two response messages $Soap_1$ and $Soap_3$ meets the *similarity threshold* , the server generates an SMP message aggregating the contents from $Soap_1$ and $Soap_3$. Figure 4.10 illustrates an SMP message, SMP_1, containing the aggregation of $Soap_1$ and $Soap_3$ messages.

The SMP protocol compares outgoing SOAP response messages in a pair-wise manner. SMP also has a user-configured time frame, denoted by t_{wait}. During this time frame, outgoing SOAP response messages are merged into an aggregated SMP message if their similarity level falls within a threshold limit. When a new request message arrives at the server, the server generates its corresponding SOAP response message and computes the new response's similarity against with the <*Common*> section of the existing merged SMP message. If the computed similarity satisfies the threshold then the new message is merged into the SMP message. If not, the existing merged SMP message is sent out and the last received response message is kept at the server for another t_{wait} period, waiting for new requests. The above aggregation steps can be repeatedly carried out. The SMP message will be also dispatched automatically after the defined period.

Back to the example, the response message $Soap_4$ for the *getStockQuote("NAB")* request from client C_3 is highly similar to the common section of SMP_1 message. Thus, $Soap_4$ is merged into SMP_1 to become SMP_2 message, which is given in Appendix C. Similarly, the responses for requests from C_4 and C_5 are also merged into the SMP_2 message. The final SMP message aggregating contents of the responses to all clients is sent out from the server S to the next-hop router R_1.

4.6.3 SMP Routing Model

The routing operation of SMP is explained in detail in this section. A network of SMP-enabled routers need to be present between the server and the client to enable the deployment of SMP. Each router along a multicast tree parses the SMP message header, partitions the client addresses into groups based on each client's next hop, replicates SMP messages if necessary, and forwards appropriate SMP messages on the next hop.

The SMP routing mechanism is described in Algorithm 1. When a router receives an SMP message, it looks in its routing table to determine the next hop for each client destination listed in the SMP header (Line 7). It then partitions the set of destinations based on the next hops (Line 9). The input SMP message that the router received is subsequently split into multiple copies depending on the number of downstream links that the router connects to (see Algorithm 2). The client list in each newly generated message header is also modified to include only those clients that will be routed through the appropriate hop. Distinctive items are also removed from the replicated messages if these items are not intended for the clients beyond the next hop. Finally, the router sends out the modified copies of the SMP messages (Line 15 of Algorithm 1).

Algorithm 1: SMP Routing Algorithm

```
1  begin
2      {$R$ is the set of next hop routers for the current SMP message $m$ }
3      {$C(m)$ is the set of clients encoded in message $m$ }
4      $R \leftarrow \emptyset$
5      foreach $C \in C(m)$ do
6          {$r$ is the next hop router for client $C$ }
7          $r \leftarrow next\_hop(C)$
8          {$C_r$ is the set of clients that have the next hop router as r}
9          $C_r \leftarrow C_r \cup C$
10         if $r \notin R$ then $R \leftarrow R \cup r$
11     foreach $r \in R$ do
12         {$m_r$ is the message to be forwarded to $r$}
13         {Call the $Message\_Splitting$ procedure}
14         $m_r \leftarrow$ Message_Splitting($m, r, C_r$)
15         Send out $m_r$ to its next hop router
16 end
```

In this way, SMP eases the network's multicast states by processing each data message on the fly along the forwarding path. If the next hop is connected to a client directly, a standard indexed SOAP unicast message is sent instead of an SMP message. On the client side, the indexed SOAP message is patched with the WSDL schema to regenerate the original SOAP message. A standard SOAP message is finally passed to the client's Application layer.

Figure 4.11 illustrates the use of SMP routing algorithm. The five clients (i.e. C_1–C_5) requests for three SOAP messages (i.e. *Soap*$_1$, *Soap*$_3$ and *Soap*$_4$). The SMP message sent out by the server S has the common part for all three messages and the distinctive part for each of them. At R_1, after parsing the header of the SMP message, the router splits the SMP message into two messages targeting two separate client groups: C_1 and C_2 via R_2, and C_3, C_4 and C_5 via R_3. Once the SMP *Message_Splitting* procedure (Algorithm 2) is applied, two SMP messages are created from the original SMP message sent by the server S. The SMP message sent to R_2 contains the union of *Soap*$_1$ and *Soap*$_3$ messages only while the SMP message sent to R_3 contains the union of *Soap*$_1$, *Soap*$_3$ and *Soap*$_4$ messages. At R_3, a unicast indexed SOAP message needs to be created from the incoming SMP message

Algorithm 2: : SMP **Message_Splitting** Procedure

```
Input: $m$: An SMP message
$r$: An intermediary router
$C_r$: The list of clients that have $r$ as next hop router
Output: $m_r$: An SMP message to forward to router $r$
1  procedure Message_Splitting do
2      foreach $C \in C(m)$ do
3          if $C \in C_r$ then
4              {$Dist(m_r)$ is the distinctive section in message $m_r$}
5              {$dist(C)$ is the distinctive part of client C in $m$}
6              $Dist(m_r) \leftarrow Dist(m_r) \cup dist(C)$
7      {$Common(m)$ is the common section of message $m$}
8      $m_r \leftarrow Common(m) + Dist(m_r)$
9      {Assign r be the next hop router for message $m_r$}
10     next_hop($m_r$) $\leftarrow r$
```

to R_3 to send the content of $Soap_4$ to C_3 because C_3 does not share paths with any other clients any more. Similarly, at the final routers (i.e. R_4, R_5 and R_6), unicast indexed SOAP messages are sent to each client individually.

Complexity Analysis

The complexity of SMP comes from the aggregation and splitting operations at SMP compatible servers and routers. Complexity directly affects SMP's performance in terms of processing time and required memory.

There are two steps in the aggregation process: XML template aggregation and XML data aggregation. Template aggregation (Definition 13 - Section 4.4.2) between two SOAP documents $Soap1$ and $Soap2$ is computed from the union and intersection of 2 sets of XML nodes. By simply matching between the elements of $\texttt{N}(Soap1)$ and $\texttt{N}(Soap2)$, the number of operations to find the union and intersection in the worst case for un-ordered sets is $|\texttt{N}(Soap1)| \times |\texttt{N}(Soap2)| = \boldsymbol{O}(n^2)$, where n is the number of distinctive XML nodes in a document tree.

Data aggregation between $Soap1$ and $Soap2$ is a major contributor to the overall complexity. Only the general case of string is considered here as other types of data can be represented as a string (e.g. binary data can be encoded in base64 as text). The primitive operations INSERT, DELETE, and SUBSTITUTE in tree edit distance [Levenshtein, 1966] are used as the unit of complexity. The most successful tree edit distance algorithm until now, which was proposed by Liang and Yokota [2005], has a complexity of $\boldsymbol{O}(d^2 \times log(d))$, where d is the length of the input string.

From the above, the complexity for aggregating both the templates and data values of $Soap1$ and $Soap2$ is $\boldsymbol{O}(n^2)+\boldsymbol{O}(nd^2log(d))$, where n is the number of XML nodes in each SOAP document and d is the data size of each XML node. The number of distinctive XML nodes in a SOAP message often has an upper bound limit as the number of data types defined in the WSDL description of a service, thus n can be considered as a constant and ignored in the complexity analysis. The complexity of the SMP aggregation operation is simplified as $\boldsymbol{O}(d^2log(d))$.

The SMP splitting operation is much less complex. The first step in this process is to find R, the set of next hop routers for all clients encoded in the incoming SMP message m. It is assumed that the routing table is an ordered set and the time it takes to complete a search for a destination is $log_2(k)$ using a binary search, where k is the number of elements in the set. For N clients , splitting will require $Nlog_2(k)$ operations. Assuming that there is an upper bound on k, the time complexity for defining the set of next hop routers is $\boldsymbol{O}(N)$. The second step in SMP splitting is to form new SMP messages where the complexity in the worst case (in which there are no common parts) is also $\boldsymbol{O}(N)$. The SMP splitting operation therefore has a complexity of $\boldsymbol{O}(N)$.

Combining the complexities of the SMP aggregation and splitting processes, it can be concluded that the proposed SMP routing algorithm has a complexity of $\boldsymbol{O}(d^2log(d) + N)$, where d is the length of a data value in a SOAP message and N is the number of clients.

4.6.4 SMP's High Level Design

Figure 4.12 summarizes the main components of SMP in a high level design. SMP lies between the SOAP engine and the Application layer. There are three main components: i) the SMP Compliant Detector; ii) the Message Handler; and iii) the Routing Adaptor. Their relationship and functionalities are explained below.

- **SMP Compliant Detector**: When a SOAP message arrives at the SOAP engine level, it will be pushed up to the SMP Compliant Detector to decide if it is a conventional SOAP message or an SMP compliant message. If it is an SMP message, this will be passed to the Message Handler; otherwise it will be passed straight to the Application layer for normal processing, thereby skipping the SMP layer.

- **Message Handler**: In case a message is received by the Message Handler, it will be analyzed by the SMP algorithm as explained in Section 4.6.3 to determine if the message needs to be split before being forwarded to the next hop routers. The message handler performs this with a help from the Routing Adaptor component.
- **Routing Adaptor**: This component is responsible for looking up the IP routing table to determine the set of next-hop routers for each client embedded in the SMP message. Additionally, the routing adaptor is in charge of routing all the outgoing SMP messages.

It is important to note that to deploy SMP in a real network, all routers in the network need to be SMP compatible. This can be done by installing an SMP software[3] on each router to enable it to interpret SMP messages. As explained in Section 4.6.2, the SOAP header in an SMP message specifies the next hop router for the message as the message's destination. Therefore, when an intermediary router receives an SMP message, it processes the message as if it is the final destination of the message. Since an SMP-compatible router operates on the application layer, it has full access to the message's envelope and parses the SOAP body to get the list of clients encoded in the SMP header and the actual payload in the SMP body.

4.7 Theoretical Analysis

The major existing SOAP performance enhancement techniques have been reviewed in Section 4.2. To date, an evaluation of these approaches is based purely on experimental results. In contrast, this section defines and thoroughly analyzed formal models of different SOAP communication schemes including unicast, multicast and the proposed SMP. Such models enable simulation results to be verified in a consistent manner and also provide a basis for comparing the different solutions in terms of average response time and total network traffic.

4.7.1 System Model

Prior to presenting the analytical models of various SOAP communication schemes, the basic system model and assumptions made in the analysis are outlined here. Table 4.1 describes the symbols and abbreviations used in this section.

[3] An SMP software is an implementation of the proposed SMP.

Symbol	**Description**
N	Number of clients.
K	Number of Web service operations.
opt_i	One in K operation responses offered by a Web service provider.
$optSize_{min}$	The minimum size of an operation response.
$optSize_{max}$	The maximum size of an operation response.
$size(opt_i)$	The size of an operation response opt_i, $optSize_{min} \leq size(opt_i) \leq optSize_{max}$.
N_i	Number of clients accessing opt_i.
s	The source node.
C	The set of clients.
φ_{query}	The mean rate of clients requests per time unit.
α	The skewness parameter of the Zipf distribution.
H	A network tree rooted at the source node s.
V	The set of nodes in the network tree H.
E	The set of links connecting any 2 adjacent nodes in H.
t_{trans}	Transmission time to transmit a message from one node to another. It is calculated as the message size divided by the bandwidth.
t_{prop}	Propagation time on each link. It is assumed to be the same for all links.
t_{link}	Total time including transmission and propagation delay for a message to pass through a link.
t_{proc}	Processing time at a node.
p_{s,c_n}	A path containing routers connecting the server to client c_n.
$\|p_{s,c_n}\|$	The number of hops between the server and client c_n.
L	A constant number representing the number of hops between the source to any client.
$n(d)$	Number of nodes at layer d in H.
$n_i(d)$	Number of nodes at layer d in multicast group tree G_i.
S_{min}	The minimum similarity between a number of SOAP messages.
S_{max}	The maximum similarity between a number of SOAP messages.
S_k	The similarity between k SOAP messages.
$aggregate\_size\{k\}$	The size of an SMP aggregated message of k SOAP original messages.
$average\_size$	The average size of a SOAP message. It is equal to $\frac{(optSize_{min}+optSize_{max})}{2}$.

Table 4.1: Legend of symbols used in theoretical models

Modeling Web Service Operations

A network of N clients that are connected to a service provider is examined in this theoretical analysis. The server (service provider) provides only one service which has K operations. Each service operation is labeled with a unique identifier, starting from opt_1 to opt_K. Operations are labeled based on their access probabilities, such that opt_1 is the most frequently accessed operation on the server, while opt_K is the least frequently accessed. N_1, N_2,..., to N_K in which $N_1 + N_2 + ... + N_K = N$, are the numbers of clients accessing operations opt_1, opt_2, ..., and opt_K respectively.

To simplify the theoretical model, it is assumed that the size of a SOAP response message, denoted as $size(opt_i)$, for one invocation of operation opt_i is uniformly distributed between $optSize_{min}$ and $optSize_{max}$. Clients generate requests to access Web service operations at a mean rate of φ_{query} queries per time unit following a Poisson distribution. To model access locality, client queries are distributed among the defined Web service operations following a Zipf-like distribution [Zipf, 1949] with a skewness parameter α. A number of studies [Barford et al., 1999; Breslau et al., 1999] report that Web requests follow a Zipf-like distribution where the number of requests to the i^{th} most popular document is proportional to $\frac{1}{i^\alpha}$, where α is a small constant skewness factor. Recent study by Oh et al. [2007] found the same also applies to Web services' access probability. The probability of an operation opt_i being requested by query Q_l can be calculated by substituting the number of operations, say K, and the operation ID i, into the Zipf distribution [Zipf, 1949].

$$Pr(opt_i \in Q_l) = \frac{1}{i^\alpha} \times \frac{1}{\displaystyle\sum_{x=1}^{K} \frac{1}{x^\alpha}} \tag{4.8}$$

$$where\ i \in 1..K\ and\ \alpha\ is\ the\ skewness\ factor.$$

Finally, since the clients make requests to the service operations at a rate of φ_{query} requests per time unit, the rate at which a client requests an operation opt_i is equal to:

$$\varphi_i = Pr(opt_i \in Q_l) \times \varphi_{query} \tag{4.9}$$

Based on Equations 4.8 and 4.9, and assuming that there are N clients in total, the number of clients, N_i, making requests for operation opt_i per time unit can be derived as follows:

$$N_i = N \times \varphi_i = N \times Pr(opt_i \in Q_l) \times \varphi_{query} = N \times \frac{1}{i^\alpha} \times \frac{1}{\sum_{x=1}^{K} \frac{1}{x^\alpha}} \times \varphi_{query} \tag{4.10}$$

Modeling the Network

Let H denote a spanning tree representing the paths from the source s to all clients C. H can be represented as follows:

$$H = \{V, E\} \tag{4.11}$$

where V is the set of nodes including the source s and the receiver set C and E is the set of links.

Except for the source and the clients, each node in the tree corresponds to a router, which is denoted r_j, where $r_j \in V$. The spanning tree H is obtained using the Open Shortest Path First (OSPF) routing algorithm [Roy, 1991]. A link is an ordered pair $(r_j, r_k) \in V \times V$ denoting a network link connecting the two routers r_j and r_k.

The transmission delay at any link is calculated by the size of the message passing through, divided by the bandwidth of the link, denoted by b_{link}. The transmission delay associated with a link r_j, r_k is denoted by $t_{trans}(r_j, r_k)$. A processing delay $t_{proc}(r_j)$ is also imposed on traffic passing through node router r_j using SMP or traditional multicast. It is assumed that from any node $r_j \in V$, there is at least one path p_{r_j,c_n} to client c_n. Client nodes $\forall c_n \in C$ are also nodes in the V set, but they are always end nodes. The total transmission cost along the path from node r_j to client c_n, $t_{trans}(r_j, c_n)$, is expressed as:

$$t_{trans}(r_j, c_n) = \sum_{(r_j, r_k) \in p_{r_j,c_n}} t_{trans}(r_j, r_k)$$

There are $|p_{s,c_n}|$ hops from the source s to a client c_n. For simplicity, these theoretical models assume that $|p_{s,c_n}|$ is constant at L for all clients, and expressed in Equation 4.12. If the numbers of hops from the source to each client are all different, it would be very complex to model the total network traffic of SMP mathematically due to not knowing where an SMP message will be split. Thus the size of an SMP message sent over a network link cannot be approximated correctly.

$$\forall c_n \in C, |p_{s,c_n}| = L \tag{4.12}$$

An example of H is depicted in Figure 4.13, where d indicates a node's layer, which is essentially the number of hops from that node to the clients.

4.7.2 Total Network Traffic

The total network traffic generated by the delivery of responses to all clients is the sum of traffic conveyed on each link along the paths from the source s to the respective client. A formula for the total traffic is developed here for each communication scheme.

Unicast Scheme

In the unicast communication scheme, where there is no mechanism to reduce the traffic sent in the network, full SOAP response messages are sent to each client as per request regardless of repetitive requests. As the result, the traffic being sent across the network for the unicast scheme is the sum of traffic generated by sending each type of operation to the clients. Therefore, we have:

$$\mathit{Total\_Traffic}\{Unicast\} = \sum_{i=1}^{K} \mathit{Traffic}\{opt_i\} \tag{4.13}$$

As defined previously, there are L hops between the server, and each client and there are N_i clients requesting operation opt_i. Therefore, the total traffic created by sending repetitive copies of SOAP responses for opt_i to N_i clients is given by the following equation:

$$\mathit{Traffic}\{opt_i\} = N_i \times L \times size(opt_i) \tag{4.14}$$

Substituting Equation 4.14 into Equation 4.13, the following function is obtained.

$$\mathit{Total\_Traffic}\{Unicast\} = \sum_{i=1}^{K} (N_i \times L \times size(opt_i)) \tag{4.15}$$

Multicast Scheme

Under the multicast communication scheme, the total network traffic is significantly reduced. This is because the server sends out only one copy of a response message to multiple clients that request the same operation. The theoretical model to compute the total network traffic for multicast scheme is given in this section. Note that it is not important which specific multicast protocol is used because the main objective is to derive the total traffic and average response time only.

In the theoretical models, there are K operations defined in a service offered by the server, so there are potentially K multicast groups set up in the network routing tree H. Each multicast group G_i corresponds to a multicast group delivering a SOAP response message to all the clients requesting the same operation. The overall network traffic generated by K multicast groups is then the aggregation of the total traffic created by each individual multicast tree. This is expressed by the following:

$$\mathit{Total\_Traffic}\{Multicast\} = \sum_{i=1}^{K} \mathit{Traffic}(G_i) \tag{4.16}$$

Next, a method to compute $\mathit{Traffic}(G_i)$ for each multicast group is defined. As explained in Section 4.7.1 “System Model”, there are N_i clients in the G_i group and every client is L hops away from the server. Consequently, G_i is an L layered multicast tree rooted at the source s and spanning N_i clients. Furthermore, it is easy to see that the number of nodes at the top layer is one (the server), the number of nodes at the bottom layer is N_i (the clients in the group), and the number of nodes at the lower layer is always greater than or equal to that at the layer above it. These observations are expressed in the following equations:

$$n_i(0) = 1 \tag{4.17a}$$

$$n_i(L) = N_i \tag{4.17b}$$

$$n_i(d+1) >= n_i(d) \tag{4.17c}$$

where $n_i(d)$ is the number of nodes in the tree G_i at layer d and $0 \le d < L$.

For a small multicast group in which the number of clients is smaller than the maximum layer in the multicast tree (i.e. $N_i < L$), we can assume that there are only N_i nodes at each layer and there is no common path among any clients. In such cases, a server needs to send unicast messages to all clients. For a larger multicast group (i.e $N_i \ge L$), since there are N_i leaf nodes in the L-layered tree, it can be assumed that there are in average $\lfloor \frac{N_i}{L} \rfloor$[4] number of nodes at each layer. These assumptions are given below:

[4] $\lfloor x \rfloor$ is the floor function of a real number x that returns the largest integer less than or equal to x.

$$n_i(d+1) = \begin{cases} N_i, & \textit{If} \quad N_i < L \\ n_i(d) + \lfloor \frac{N_i}{L} \rfloor, & \textit{If} \quad N_i \geq L \end{cases} \quad where \quad 0 \leq d < L \tag{4.18}$$

In the traditional multicast case [Oliveira et al., 2005; Zhang et al., 2002], the size of the message passing through the edges of a multicast group tree remains the same throughout the sub-tree, which equals to $size(opt_i)$ for the G_i sub-tree. Traffic in a multicast tree can be calculated as the sum of traffic on each layer. The traffic incurred on each layer is then equal to the message size multiplied by the number of nodes at that layer. The total traffic in G_i is described by the following equation.

$$\textit{Traffic}(G_i) = \sum_{d=1}^{L} \textit{traffic}(d_{G_i}) = \sum_{d=1}^{L} n_i(d) \times size(opt_i) \tag{4.19}$$

Finally, an expression for *Total_Traffic*{*Multicast*} that is the total traffic generated in the network by using the multicast scheme, is derived by combining Equations 4.19 and 4.16.

$$\textit{Total\_Traffic}\{Multicast\} = \sum_{i=1}^{K} \sum_{d=1}^{L} n_i(d) \times size(opt_i) \tag{4.20}$$

SMP Scheme

A model for formalizing the total network traffic in the SMP communication scheme is presented here. For simplicity, it is assumed that the similarity threshold is met by all N clients, meaning that all the responses to N clients can be aggregated and sent together in one SMP message.

An SMP routing tree is also assumed to be an L-layered spanning (H) tree. H is rooted at s and spans N clients. Similar to a multicast sub-tree G_i discussed above, the H tree possesses the following charateristics.

$$n(0) = 1 \tag{4.21a}$$

$$n(L) = N \tag{4.21b}$$

$$n(d+1) = \begin{cases} N, & \textit{If}\, N < L \\ n(d) + \lfloor \frac{N}{L} \rfloor, & \textit{If}\, N \geq L \end{cases} \tag{4.21c}$$

where $n(d)$ is the number of nodes in H at layer d and $0 \leq d < L$.

A formula approximating the size of an aggregated SMP message from k original SOAP response messages is proposed here. For simplicity, the similarity between k messages is assumed to be uniformly distributed between S_k^{min} and S_k^{max}. A different distribution of message similarity can also be used , however it requires higher complexity to model. An SMP message is composed of a common section of k ordinary messages and a distinctive section that contains unique parts of each message. As portrayed in Figure 4.14, the darker color section in the diagram represents the common section (in the middle).

Assuming that all k messages have the same size, the size of the common section is then proportional to the similarity of the messages, denoted by S_k. The different part (in lighter color) of a message is therefore proportional to $(1 - S_k)$. As a result, the total size of the aggregated message can be computed approximately as below.

$$aggregate\_size\{k\} = (S_k + k(1 - S_k)) \times msg\_size = (k - (k-1)S_k) \times msg\_size \tag{4.22}$$

Assuming that there is a uniform distribution of message size between $optSize_{min}$ and $optSize_{max}$, the average size of a SOAP response message, msg_size, is given by the following equation.

$$msg\_size = \frac{optSize_{min} + optSize_{max}}{2}$$

With the assumptions on the tree's structure given in Equations 4.21a, 4.21b and 4.21c, the average number of clients included in an SMP message passing through each node at layer d can be given as $\lfloor \frac{N}{n(d)} \rfloor$. After substituting k with $\lfloor \frac{N}{n(d)} \rfloor$ in Equation 4.22, the total traffic sent across a link at layer d in the SMP tree becomes:

$$aggregate\_size\{\frac{N}{n(d)}\} = (\lfloor \frac{N}{n(d)} \rfloor - (\lfloor \frac{N}{n(d)} \rfloor - 1)S_{\frac{N}{n(d)}}) \times msg\_size \tag{4.23}$$

Generalizing Equation 4.23 for all incoming links from the source s to all nodes that are d hops away from s, allows us to obtain the following formula:

$$Traffic(d_{SMP}) = \begin{cases} n(d) \times (\lfloor \frac{N}{n(d)} \rfloor - (\lfloor \frac{N}{n(d)} \rfloor - 1) \times S_{\frac{N}{n(d)}}) \times msg\_size, & if \quad n(d) < N \\ \sum_{i=1}^{K} N_i \times size(opt_i), & if \quad n(d) = N \end{cases} \tag{4.24}$$

A separate formula for the cases when $n(d) = N$ is needed because a total of N unicast messages of different sizes will be sent.

The total traffic generated in the SMP tree is the sum of all traffic created at each layer. Based on Equation 4.24, an expression to compute the overall network traffic in an SMP routing tree can be obtained as follows:

$$\mathit{Traffic}\{SMP\} = \sum_{d=1}^{L} \begin{cases} n(d) \times (\ \lfloor \frac{N}{n(d)} \rfloor - (\lfloor \frac{N}{n(d)} \rfloor - 1) \times S_{\frac{N}{n(d)}}\ ) \times msg\_size, & if \quad n(d) < N \\ \sum\limits_{i=1}^{K} N_i \times size(opt_i), & if \quad n(d) = N \end{cases} \tag{4.25}$$

By not listing out the formula for $aggregate\_size\{\frac{N}{n(d)}\}$, a more compact version of the above equation is shown below:

$$\mathit{Traffic}\{SMP\} = \sum_{d=1}^{L} \begin{cases} n(d) \times aggregate\_size\{\frac{N}{n(d)}\}, & if \quad n(d) < N \\ \sum\limits_{i=1}^{K} N_i \times size(opt_i), & if \quad n(d) = N \end{cases} \tag{4.26}$$

4.7.3 Average Response Time

Apart from ensuring that the network traffic is reduced, the response time experienced by the clients is also an important performance criterion. The response time is the time from when the server receives a request from a client to the time the client receives the response message. The lower an average response time is, the more responsive the system is to client requests. SMP is well suited to applications where there are many similar simultaneous requests to the server. In this model, it is assumed that all N clients make queries to the server at the same time. The average response time for any communication scheme is given by the following equation:

$$Average\_Response\_Time = \frac{\sum\limits_{n=1}^{N} response(c_n)}{N} \tag{4.27}$$

The response time experienced by an individual client is composed of the delay to convey the response message in the network plus any overhead at the server and at intermediate routers if they exist. The total delay then comprises of the sum of transmission delay and propagation delay across the path from the source to the client. The transmission time, denoted by t_{trans}, of a message passing through a link is computed by dividing the size of the message, $size(opt_i)$, by the link bandwidth. It is assumed that the bandwidth on each link in the network is the same and is equal to b_{link}. The propagation delay, denoted by t_{prop}, is the time required for a message to travel from one point to another, and is assumed to be constant across all links. The time it takes to convert an indexed SOAP message to a standard SOAP message on the client side is not included in the average response time model because this time is considered considerably smaller than propagation or transmission delay. The response time experienced by each client is given by the following formula:

$$\begin{aligned} response(c_n) &= t_{overhead}(c_n) + delay(c_n) = t_{overhead}(c_n) + \sum_{e \in p_{s,c_n}} (t_{prop} + t_{trans}(e)) \\ &= t_{overhead}(c_n) + \sum_{e \in p_{s,c_n}} (t_{prop} + \frac{size(opt_i)}{b_{link}}) \end{aligned} \tag{4.28}$$

Unicast Scheme

With the unicast scheme, the processing overheads at the server and the routers are negligible and therefore ignored in our model (i.e. $t_{overhead}(c_n)\{unicast\} = 0$). N clients can be split into K groups of clients. Each group has $N_1, N_2, ...,$ and N_K clients. All clients in one group receive the same service operation response (i.e. a group with N_i clients receives a response for opt_i). Also because of the assumption that there are L hops between the source and any client, the number of links from the source s to a client c_n equals to L. Applying these to Equations 4.27 and 4.28, the average response time for the unicast scheme is computed as follows:

$$Average\_Response\{unicast\} = \frac{\sum_{i=1}^{K}(N_i L(t_{prop} + \frac{size(opt_i)}{b_{link}})}{N} \tag{4.29}$$

Multicast scheme

In the traditional multicast scenario, when a client sends a request to the server, the server will not respond to it promptly but rather it will wait for a number of other requests from other clients who are requesting the same service operation. This is because the server needs to get a reasonable number of the same requests from the clients before sending a multicast message out. This waiting period, denoted by $t_{procServer}(multicast)$, is significant and does not exist in the unicast scheme, but is included in the average response time for the multicast scheme.

Under the multicast scheme, when a message arrives at an intermediary router the router needs to process the multicast address and determines if extra copies of the message have to be created to send to branching routers. Therefore, there may be a small processing delay incurred. The average processing time at an intermediate router for multicast is denoted by $t_{procRouter}(multicast)$. The total overhead time for sending a message to a client, denoted as $t_{overhead}(c_n)$, is then the sum of the server processing time and the total router overhead over the path reaching the client, which is expressed below:

$$t_{overhead}(c_n)\{multicast\} = t_{procServer}(multicast) + L \times t_{procRouter}(multicast) \tag{4.30}$$

In addition, the transmission delay to deliver a multicast message to a client is the same as to deliver a unicast message because the number of hops from the source to any client is assumed to be the same and the size of the message is the same in either unicast or multicast scenario. By substituting Equation 4.30 into Equations 4.27 and 4.28, the average delay experienced by each client for the multicast case can be expressed as:

$$\begin{aligned} Average\_Response\{multicast\} &= \frac{1}{N}\sum_{i=1}^{K}\sum_{n=1}^{N_i}(\ t_{procServer}(multicast) \\ &+ L(\ t_{procRouter}(multicast) \\ &+ t_{prop} + \frac{size(opt_i)}{b_{link}}\ )\ ) \end{aligned} \tag{4.31}$$

SMP scheme

In SMP, there is an extra overhead to measure the similarity between SOAP messages when clustering the messages into groups and to generate SMP messages based on the common and different parts of the messages. This overhead at the server is denoted by $t_{procServer}(SMP)$. The value of this variable depends on the number of outgoing response messages and work required to measure message similarities.

The overhead at intermediary routers in the SMP case is also similar to that in the traditional multicast case. However, $t_{procRouter}(SMP)$ may be greater than $t_{procRouter}(multicast)$ because more time is required to split SMP messages to branching routers. Similar to multicast, the total overhead time, $t_{overhead}(c_n)$, for sending a message to a destination in the SMP case is expressed as follows.

$$t_{overhead}(c_n)\{SMP\} = t_{procServer}(SMP) + L \times t_{procRouter}(SMP) \tag{4.32}$$

The total transmission time of all messages in the SMP case is computed by dividing the total transmitted traffic by the network bandwidth. The total transmitted traffic here is the sum of traffic sent to all clients. In a case of aggregated messages passing through a layer (i.e. $n(d) < N$), the total transmitted traffic is computed by multiplying the traffic generated on that layer, as given in Equation 4.26, by the average number of clients, denoted as $\lfloor \frac{N}{n(d)} \rfloor$, encoded in each message. In a case when unicast messages are sent at a layer, the total transmitted traffic on that layer is the same as the one given in Equation 4.26 (in $n(d) = N$ case). Therefore, the total transmission time experienced by all clients is given by the following formula:

$$t_{trans}\{SMP\} = \frac{1}{N \times b_{link}} \sum_{d=1}^{L} \begin{cases} \lfloor \frac{N}{n(d)} \rfloor \times n(d) \times aggregate\_size\{\frac{N}{n(d)}\}, & if \quad n(d) < N \\ \sum\limits_{i=1}^{K} N_i size(opt_i), & if \quad n(d) = N \end{cases} \tag{4.33}$$

Combining Equations 4.27, 4.28, 4.32 and 4.33, the formula for computing SMP's average response time is given by the following:

$$Average\_Response\{SMP\} = t_{procServer}(SMP) + L(\ t_{procRouter}(SMP) + t_{prop}\ ) \tag{4.34}$$

$$+\frac{1}{N \times b_{link}} \sum_{d=1}^{L} \begin{cases} \lfloor \frac{N}{n(d)} \rfloor \times n(d) \times aggregate\_size\{\frac{N}{n(d)}\}, & if \quad n(d) < N \\ \sum\limits_{i=1}^{K} N_i size(opt_i), & if \quad n(d) = N \end{cases} \tag{4.35}$$

4.8 Simulation and Results

In order to evaluate the performance of the proposed method, extensive experimentation has been performed. This section describes the simulation model and summarizes the results obtained from simulation of the proposed SOAP multicast protocol, traditional multicast, and unicast.

4.8.1 Experimental Setup

It is important to choose a valid test data domain because it needs to be suitable for the use of SMP. SMP is best suited in high-transaction services where the server needs to send many syntactically similar response messages to a large number of clients in a short period of time. Hence, data obtained from the Australian Stock Exchange has been chosen for testing the Stock Quote application to evaluate the performance of SMP.

The OMNeT++ [Varga, 2006] simulation program was used to randomly generate different hierarchical network topologies with different numbers of network layers to carry out the experiments. Figure 4.15 illustrates a simple network of 20 clients used for testing. For

Parameter	**Default**	**Range**
Number of simulation runs per data point	20	
Number of queries in a simulation period	300	20–500
Number of clients (N)	50	10–200
Number of service operations (K)	10	5–50
Size of a SOAP response message (K)	30Kb	20–50Kb
Number of routers between the source and a client (tree depth)	10	5–15
Bandwidth of a link (b_{link})	1.5Mbps	
Propagation time through a link (t_{prop})	1 ms	
Processing time at routers in multicast scheme ($t_{proc}multicast$)	0.1 ms	
Processing time at routers in SMP scheme ($t_{proc}SMP$)	0.2 ms	
Similarity threshold	0.7	0.3–1
Zipf parameter (α)	1	0.4–1.8

Table 4.2: Simulation Parameters

each network topology setup, tests of SMP, traditional multicast (which uses IP multicast), and unicast communication schemes were performed. The simulation model consisted of N clients making queries to K operations. Queries were made, based on a Zipf distribution [Zipf, 1949], among all operations offered by the Web service server. The message size and the number of clients were varied in different tests. For each test, 20 experimental runs were performed and the result presented is the average of these runs. The simulation parameters and their values are listed in Table 4.2.

4.8.2 Experimental Results

This section examines the performance of SMP by measuring the total network traffic and the average response time of each communication scheme. The total network traffic is the sum of all the sizes of messages sent to clients. The average response time is the time from when the server sends a response message until the message reaches its targeted client. It is computed by dividing the total delay experienced by the number of clients. Response time includes propagation and transmission delays on each link and processing delays at the server and intermediary nodes.

Total Network Traffic

Figure 4.16 shows the simulation results giving the total network traffic for the three communication schemes. The unicast scheme produces the greatest volume of traffic, which is proportional to the number of receivers. The traditional multicast protocol represents an improvement of around 30% over unicast. SMP reduces traffic by up to 50% over unicast. With a small network (10 to 50 clients), the reduction in traffic of SMP over unicast or multicast is already noticeable, up to 35%. With larger networks (100 to 200 clients) SMP's performance gain over unicast in traffic becomes even more significant, between 45–50%. Comparing SMP to multicast, the difference in traffic generated is not noticeable with small networks of 10 or 20 clients. When the client numbers increase to over 100 clients, SMP outperforms traditional multicast by around 25%. In general, the larger the network is, the higher performance SMP offers in terms of the reduction of network traffic.

Average Response Time

The graph in Figures 5.8 displays the simulation results for the average response time when different SOAP communication schemes are used. The unicast method has the lowest average response time at approximately 86ms for networks with 100 clients and 116ms for networks with 200 clients. Corresponding results for multicast and SMP are 103ms (100 clients), 153ms (200 clients) and 114ms (100 clients), 175ms (200 clients) respectively. That is, the traditional multicast protocol is about 1.3 times slower than unicast.

SMP's weakness of having higher processing overhead at the server and intermediate nodes causes the average response time of SMP to be slightly higher than that of the multicast and unicast method. However, the delay penalty is negligible when the number of clients is small (under 40 clients). When the network size increases to 100 clients, SMP experiences 1.5 times higher response time than unicast and 1.15 times higher than multicast.

Payload Size Factor

In this section, the performance of SMP is analyzed in two scenarios: small messages (0.6–1Kb) and large messages (100–300Kb).

In the small message scenario, the message schema and other parameters are unchanged. Only the message payload is changed, with size ranging from 0.6 to 1Kb. Figures 4.18(a) and 4.18(b) show the results for total network traffic and average response time for different routing protocols in this scenario. As shown by the graph, the performance gain of SMP over multicast is marginal, under 10% improvement for networks of under 80 clients. Compared to unicast, SMP can reduce total traffic by only 40% in the small message scenario, compared to 50% in the normal scenario presented above. Since SMP messages include extra XML encoding for grouping common and distinctive sections, an SMP message aggregating SOAP messages with small payload does not result in a significant size reduction compared to the combined size of all individual messages. The average response time of SMP in the small message case is slightly smaller than that in the medium message case because of less work involved in the measurement of similarity between payload data.

Figure 4.19(a) displays the total network traffic of SMP against multicast and unicast in the large message scenario. In this scenario, large payload data was simulated for the same service schema. Message sizes ranged from 100 to 300Kb. SMP shows better performance with large payload size compared to other scenarios. As shown in Figure 4.19(a), even with small networks (under 50 clients) SMP outperforms unicast by 60%. Furthermore, a close to 70% reduction in network traffic can be achieved for large networks (more than 150 clients). In the large message scenario, SMP performs slower than in other scenarios by around ten miliseconds — 1.6 times higher in response time, which is not significant in many applications.

4.8.3 Validation of the Results

In this section, the experimental results are compared to the analytical results presented in Section 4.7. Figures 4.20 and 4.21 show both the analytical and experimental results in total network traffic and average response time for all three communication schemes. Table 4.3 outlines the values of the key parameters used when obtaining the graphs.

Parameter	**Value**
$size_{msg}$	35Kb
b_{link}	1.5Mbps
t_{prop}	5ms
$size(opt_i)$	35Kb
$t_{procServer}(multicast)$	0.5ms
$t_{procRouter}(multicast)$	0.1ms
$t_{procServer}(SMP)$	0.6ms
$t_{procRouter}(SMP)$	0.2ms

Table 4.3: Assumptions of parameters used to obtain the theoretical results

As seen from Figure 4.20, the simulation results generally follow the patterns predicted by the analytical results. However, there is a large difference between the experimental and analytical results of multicast for networks over 100 clients. This is due to assumptions made (for simplicity) about the fixed network topology in the theoretical analysis (see Section 4.7.2). In the simulations, generated network topologies did not strictly follow the assumption on the number of nodes in each layer because in the experiments, the network topologies were randomly created. Therefore, some discrepancies between the analytical and simulated results are expected.

Similarly, the average response times in simulations closely follow the expected trends in the analytical results as shown in Figure 4.21. These results prove that SMP outperforms both unicast and traditional multicast methods in terms of total network traffic in both experimental and theoretical analysis.

4.9 Discussion

SMP is a SOAP multicast protocol that makes use of the similarity of SOAP messages to reduce network traffic significantly and to increase overall SOAP performance. In addition, SMP uses a combination of both multicast and unicast concepts to actually 'multicast' messages without using multicast addresses. As discussed in the experimental results section, SMP reduces the consumption of network bandwidth by i) around 20% for small networks with small data payload (0.6–1Kb); ii) by over 40% for networks with medium-size messages (20–50Kb); and iii) by 50–70% for large networks with large-size messages (100–300Kb). SMP's performance gain in network traffic improvement is proportional to the number of clients in the multicast group.

The improvement in total network traffic of SMP over unicast is obvious because under SMP, messages of similar content are grouped and sent together. Therefore, duplicate messages are avoided and more bandwidth can be saved. SMP can reduce further traffic than conventional IP multicast because SMP can 'multicast' SOAP messages of slightly different contents as well as identical messages while IP multicast can send only same messages. Common parts (templates or data values) in different outgoing SOAP messages are efficiently reused and only one copy of the common parts is sent out. In addition, SMP employs an indexing technique that only compact representations of SOAP documents are sent, therefore the total size of messages can be reduced further.

SMP can support a very large number of receivers because there is no limit on the number of addresses that could be embedded in a SOAP message (SMP header), while the size of Xcast (a small group multicast protocol, see Section 4.3.1) packet headers is limited to the size of an IP header, so the number of receivers that can be stored inside an Xcast header is limited. SMP addresses the scalability problem of Xcast. Placing client addresses inside SOAP messages is also beneficial for reliability since the senders always have the addresses of the receivers for retransmission purposes. It is important to note that no state maintenance is required at forwarding routers under SMP. In contrast, under IP multicast each router on multicast trees needs to maintain a list of clients that belong to the multicast group.

A drawback of SMP is that it requires additional processing time at each router, to perform unicast table lookups, to parse the SMP header, and to split SMP messages into multiple copies. Thus, SMP produces higher average response time than unicast. SMP provides a multicast solution with minimal usage of network bandwidth and a low delay trade-off. It is especially suitable for applications that do not have strong constraints in delay, as well as for wireless environments (ad-hoc networks, wireless LANs and wide-area cellular networks) which are scarce in bandwidth.

4.10 Summary

This chapter presented SMP, which is a new multicast protocol for Web services to efficiently send similar SOAP messages to a large number of clients. The contribution of this chapter is three-fold. Firstly, a novel algorithm for measuring the similarity of SOAP messages was presented. Using this similarity measurement between outgoing SOAP messages, servers can determine if these messages have the XML structure or payload values in common. Secondly, a new similarity-based multicast routing protocol (SMP) for SOAP is proposed to take advantage of common message structures and values among response messages. SMP sends the contents of multiple similar responses in one aggregated message, called an SMP message, instead of sending each of them individually, thus significantly reducing network traffic.

Finally, a SOAP indexing technique is proposed to enable fast merging and splitting of SMP messages. Each data type definition in a WSDL service description is indexed with a unique identifier. Each XML node in a SOAP message is also labeled with a position, and is presented in a compact form that includes: (1) the node data type ID referenced back to the WSDL document, (2) the position of the node in the SOAP message and (3) the node value. The indexed representation of a SOAP message not only reduces message size due to the omission of full XML tag names, but also leverages the organization of common and distinctive data in SMP messages. To the best of our knowledge, SMP is the first similarity-based SOAP multicast protocol that has been proposed in the Web services literature.

SMP's main advantage is that it reduces the traffic generated over a network by 50–70% compared to unicast in large networks (over 100 clients) with large payload (over 100Kb).

The performance trade-off of SMP is its higher response time. However SMP's performance penalty in response time is relatively small (1.5 times slower) compared to its performance gain in network traffic (3 times less traffic), when being compared with unicast. The use of SMP is justifiable for Web service applications that are not time critical, are constrained in bandwidth and deal with large amount of data.

```
<soap:Envelope><soap:Header>
  Next-hop router's address
 </soap:Header>
 <soap:Body>
  <SMP:Header>
   <c id='1', uri='URI1'/>
   <c id='2', uri='URI2'/>
  </SMP:Header><SMP:Body>
   <Distinctive>
    <DPart>
      <cPos cIds ='1,2' pos='0'/>
      <part refer='p1'/>
    </DPart>
    <DPart>
      <cPos cIds ='1' pos='1'/>
      <part>
       <v eRef= 'AOSQ' val='2'/>
      <part/>
    </DPart>
    <DPart>
      <cPos cIds ='2' pos='1'/>
      <part>
       <v eRef= 'AOSQ' val='3'/>
      <part/>
    </DPart>
    <DPart>
      <cPos cIds ='1,2' pos='2.1'/>
      <part refer='p2'/>
    </DPart>
    <DPart>
      <cPos cIds ='1' pos='2.2'>
      <cPos cIds ='2' pos='2.3'>
      <part refer='p3'>
    </DPart>
    <DPart>
      <cPos cIds ='2' pos='2.2'>
      <part id='p4'>
       <v eRef='SQ' val=''/>
       <v eRef='Sbl' val='WIL'/>
       <v eRef='QI' val=''/>
       <v eRef='P' val='1.29'/>
       <v eRef='LU' val='24/01/2007 10:45am'/>
      </part>
    </DPart>
   </Distinctive>
 <Common>
     <part id='p1'>
       <v  eRef= 'SQR' val=''/>
     </part>
     <part id='p2'>
       <v  eRef= 'SQ' val=''/>
       <v  eRef= 'Sbl'  val='NAB'/>
       <v eRef='QI' val=''/>
       <v eRef='P' val='28.56'/>
       <v eRef='LU' val='24/01/2007 10:15am'/>
     </part>
     <part id='p3'>
       <v eRef='SQ' val=''/>
       <v eRef='Sbl' val='BHP'/>
       <v eRef='QI' val=''/>
       <v eRef='P' val='24.52'/>
       <v eRef='LU' val='24/01/2007 10:45am'/>
     </part>
   </Common>
  </SMP:Body>
 </soap:Body>
</soap:Envelope>
```

Figure 4.10: SMP_1: An SMP message aggregating $Soap_1$ (Figure 4.2) and $Soap_3$ (Figure 4.4) messages

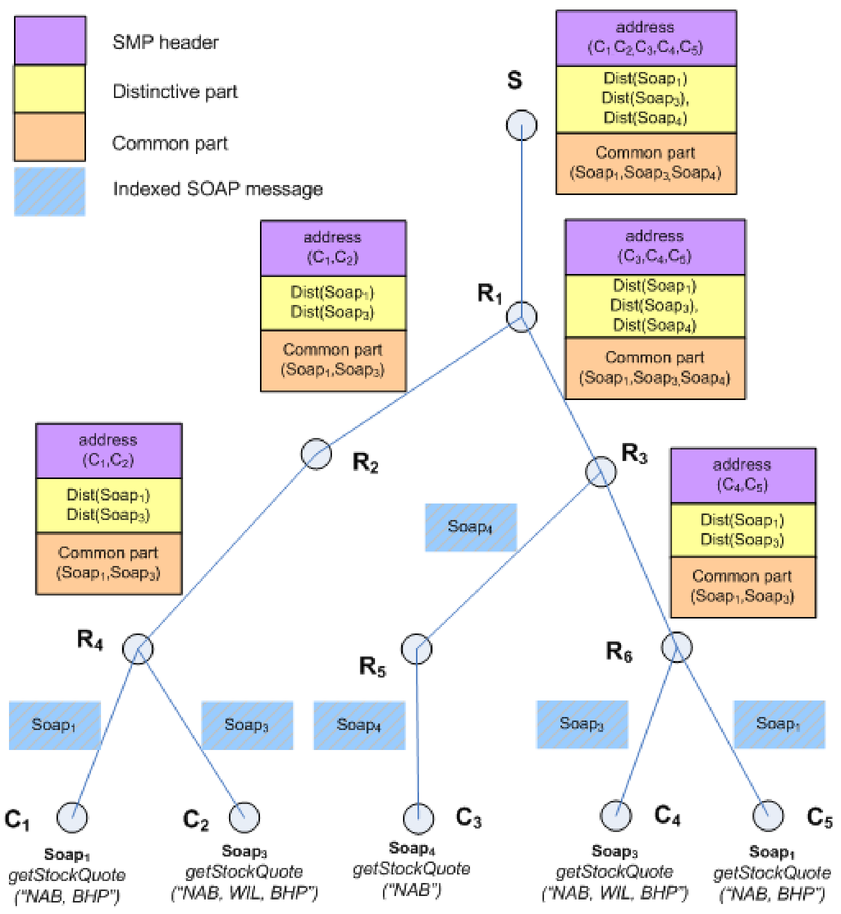

SMP header
Distinctive part
Common part
Indexed SOAP message
S
address (C_1,C_2,C_3,C_4,C_5)
Dist($Soap_1$) Dist($Soap_3$), Dist($Soap_4$)
Common part ($Soap_1$,$Soap_3$,$Soap_4$)
R_1
address (C_1,C_2)
Dist($Soap_1$) Dist($Soap_3$)
Common part ($Soap_1$,$Soap_3$)
address (C_3,C_4,C_5)
Dist($Soap_1$) Dist($Soap_3$), Dist($Soap_4$)
Common part ($Soap_1$,$Soap_3$,$Soap_4$)
R_2
R_3
address (C_1,C_2)
Dist($Soap_1$) Dist($Soap_3$)
Common part ($Soap_1$,$Soap_3$)
$Soap_4$
address (C_4,C_5)
Dist($Soap_1$) Dist($Soap_3$)
Common part ($Soap_1$,$Soap_3$)
R_4
R_5
R_6
$Soap_1$
$Soap_3$
$Soap_4$
$Soap_3$
$Soap_1$
C_1
C_2
C_3
C_4
C_5
$Soap_1$ getStockQuote ("NAB, BHP")
$Soap_3$ getStockQuote ("NAB, WIL, BHP")
$Soap_4$ getStockQuote ("NAB")
$Soap_3$ getStockQuote ("NAB, WIL, BHP")
$Soap_1$ getStockQuote ("NAB, BHP")

Figure 4.11: An example showing how SMP routing mechanism works

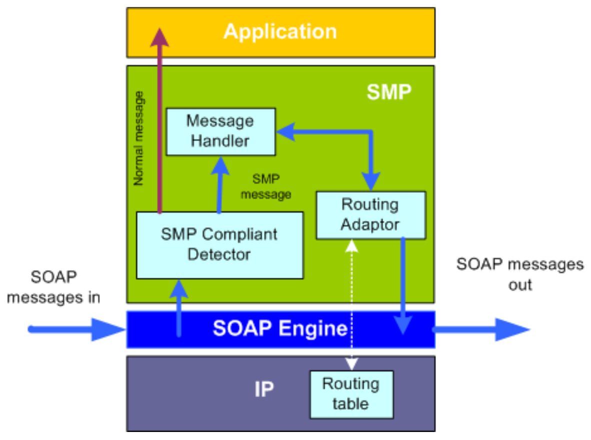


Figure 4.12: SMP's high level design

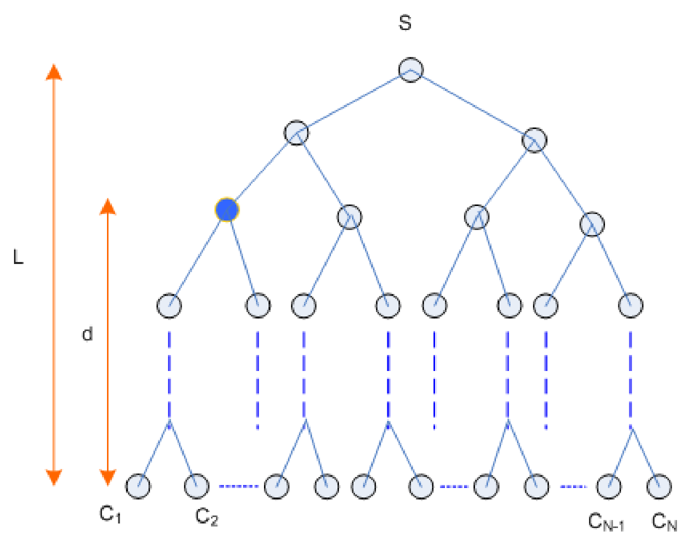


Figure 4.13: The routing tree used for developing theoretical models

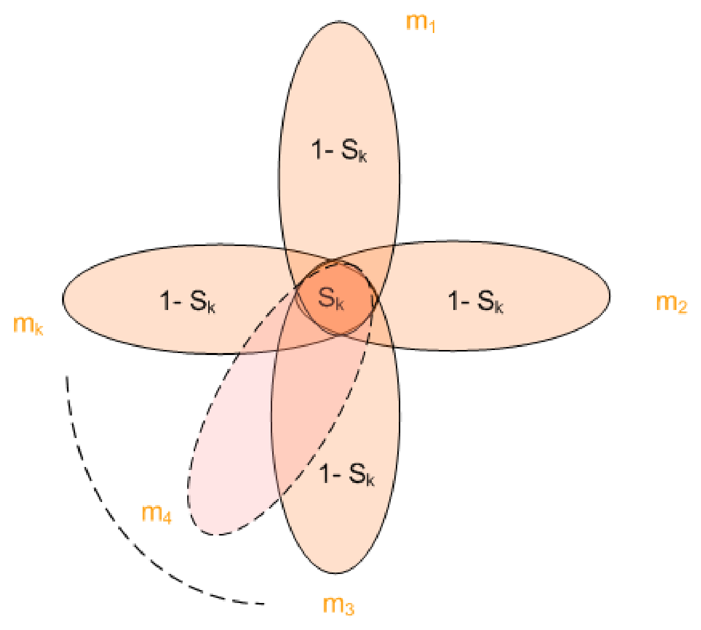


Figure 4.14: A model to approximate the size of an SMP aggregated message of k SOAP response messages

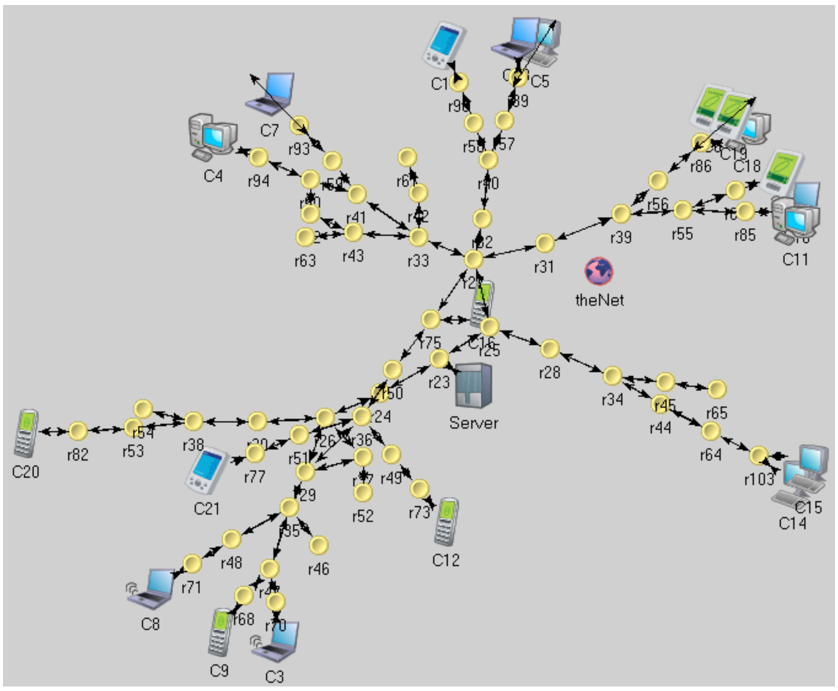


Figure 4.15: An example of a simple simulated network

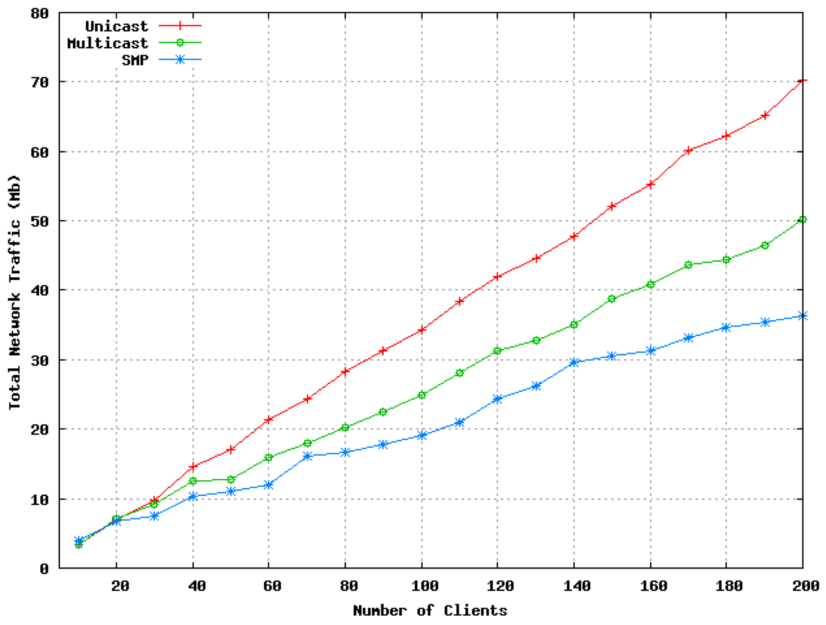


Figure 4.16: Total network traffic for SMP, multicast and unicast routing protocols with medium messages of 20–50Kb.

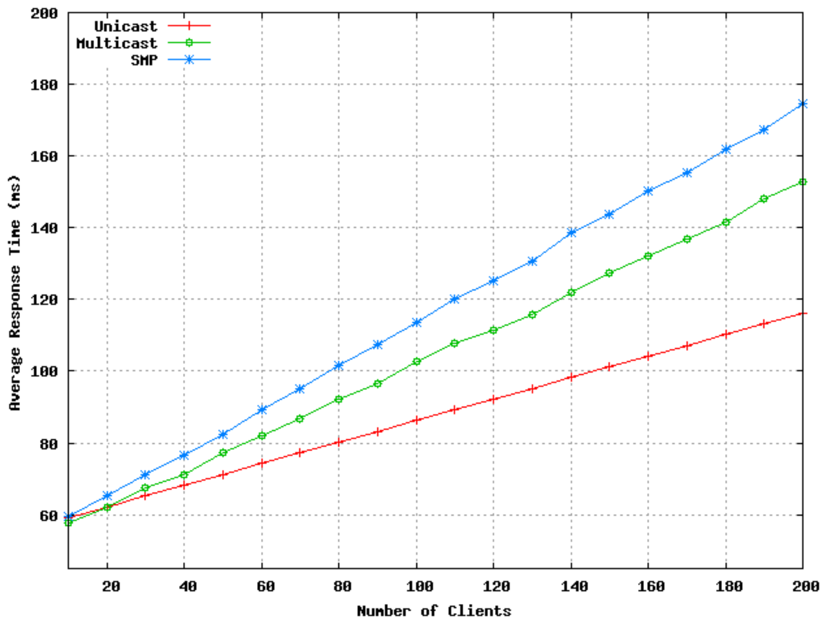
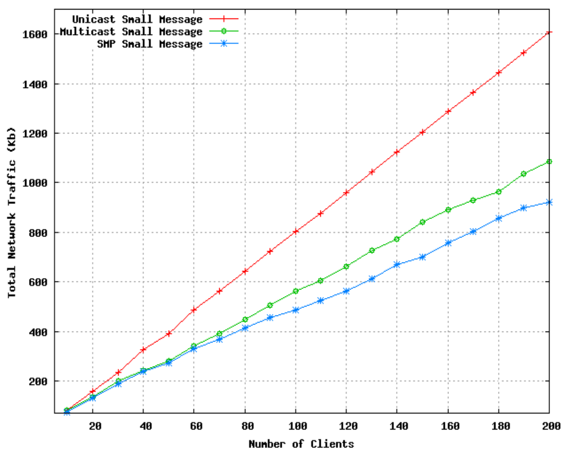
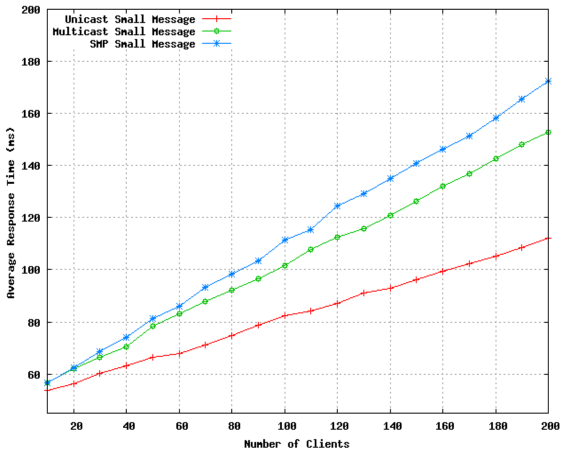


Figure 4.17: Average response time for SMP, multicast and unicast routing protocols with medium messages of 20–50Kb.

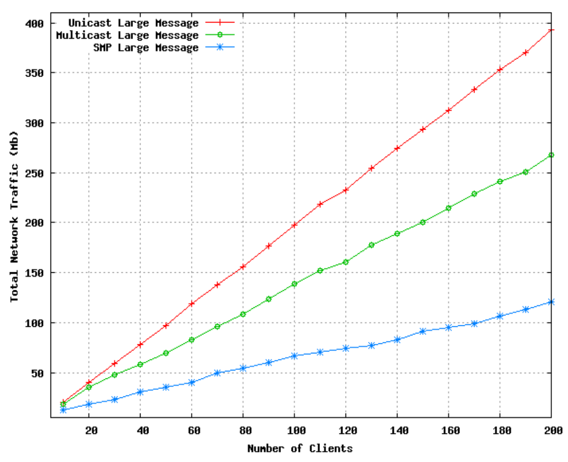


(a) Total network traffic

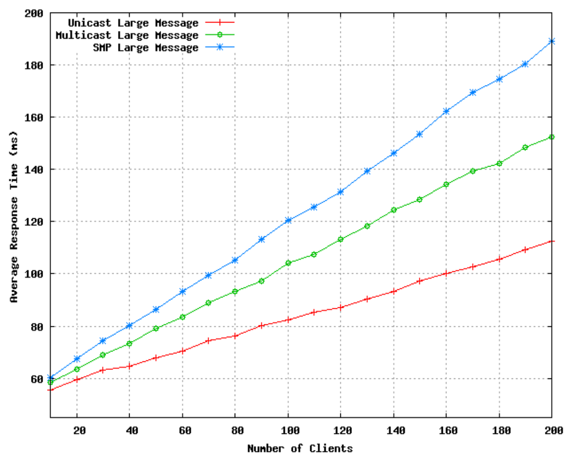


(b) Average response time

Figure 4.18: Total network traffic and average response time for SMP, multicast and unicast routing protocols with small messages of 0.6–1Kb.



(a) Total network traffic



(b) Average response time

Figure 4.19: Total network traffic and average response time for SMP, multicast and unicast routing protocols with large messages of 100–300Kb.

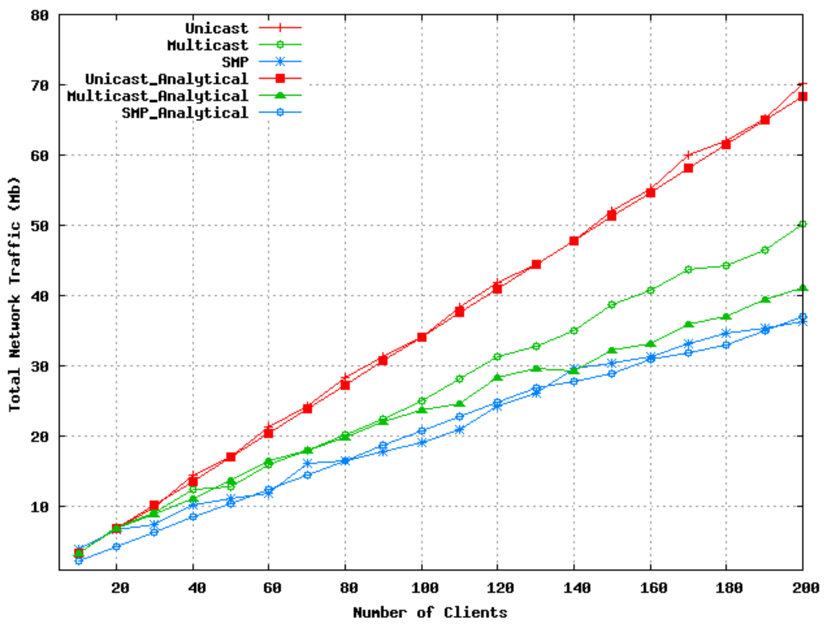


Figure 4.20: Analytical total network traffic analysis for different routing protocols.

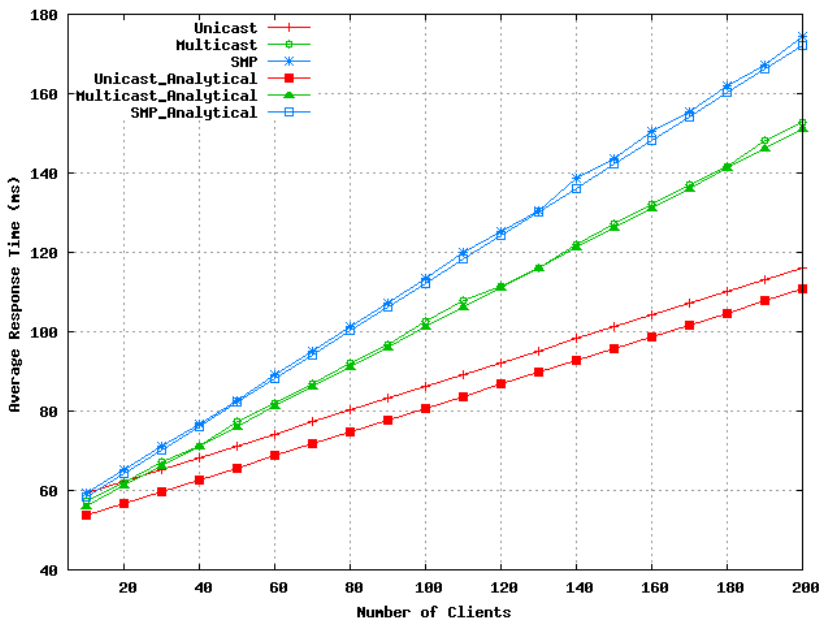


Figure 4.21: Analytical average response time analysis for different routing protocols.

Chapter 5

SMP's Extension for Network Traffic Optimization*

This chapter presents a traffic-constrained SOAP multicast routing protocol, called tc-SMP[1], which is an extension of the similarity-based SOAP multicast protocol (SMP) proposed in the previous chapter. Tc-SMP offers improvements over SMP in terms of network traffic size by using its own routing protocol instead of the shortest path algorithm that conventionally sends messages between nodes along shortest paths first. Tc-SMP looks at the network optimization aspect of the SMP protocol and proposes alternative message delivery paths that minimize the number of bytes transmitted over the network. Two tc-SMP algorithms, *greedy* and *incremental* approaches, are proposed and compared for their efficiency and performance advantages over SMP. Simple heuristic methods are also proposed for incorporating with the two tc-SMP algorithms to improve results. Theoretical analysis is also presented to show the differences in total traffic and average response time between tc-SMP and SMP. From extensive experiments, it is shown that tc-SMP achieves a 25% reduction in total network traffic compared to SMP, with a trade-off of 10% increase in average response time. Compared to conventional unicast, bandwidth consumption can be reduced by up to 70% using tc-SMP and 50% using SMP.

* Preliminary versions of the work presented in this chapter have been published in [Phan et al., 2007b] and [Phan et al., 2007a]. [1] In earlier publication of this work, tc-SMP was called eSMP [Phan et al., 2007b].

5.1 Motivation

SMP, which was introduced in Chapter 4, uses the Open Shortest Path First (OSPF) protocol to route messages. This protocol is based on Dijkstra's algorithm [Cormen et al., 2001], and it routes a message using the shortest path from a source to a destination. Therefore, the routing path selected by such an algorithm may not be the best in terms of traffic optimization. It is important to note that two nodes of a network are connected with multiple paths in most cases.

The routing protocol plays a critical role in SMP. In particular, the more similar SOAP messages can be combined over specific links the more network bandwidth can be saved. However when OSPF is used, some SOAP messages that are very similar in content and follow the shortest paths, there may not be many common links shared by these messages. By restricting ourselves to the OSPF routing paths, we may miss other feasible paths that may result in further reduction in the total traffic. Using other paths may permit the aggregation of SOAP messages with a higher level of similarity and their transmission along more common links.

This chapter describes an extension of SMP, called tc-SMP, which includes an efficient routing algorithm for delivering SMP messages. Simulations have shown that tc-SMP can reduce network traffic, when compared to SMP, by more than 25% with a small increase in response time (namely 10%). In addition, tc-SMP can reduce the traffic by up to 70% compared to unicast, and 40% compared to traditional multicast, the corresponding results for SMP are 60% and 30% respectively.

The rest of this chapter is organized as follows. The next section discusses related work on QoS-based routing. The tc-SMP routing algorithm is then presented in Section 5.4. Analytical comparisons between the tc-SMP algorithms and the SMP algorithm are presented in Section 5.4.5. Performance experiments and results are detailed in Section 5.6. Afterwards, the experimental results are analyzed in the Discussion section to highlight the strengths and weaknesses of tc-SMP. The chapter concludes with a summary that remarks on tc-SMP and its benefits.

5.2 Related Work

This section gives an overview description of QoS-based routing and discusses existing popular QoS-based routing solutions in multicast. In recent years, there has been a rapid growth in multimedia and real-time applications such as video teleconferencing, real-time remote control systems, and interactive distributed games. This evolution of digital technologies has arisen a need to satisfy certain quality-of-service (QoS) requirements for individual customers who are willing to pay more for better services such as faster response time or higher bandwidth. Service providers may have other QoS requirements that relate to the whole session, such as minimal transmission cost. By adding such constraints to the original problem of routing, the problem is re-formulated as a QoS-constrained routing problem.

5.2.1 QoS-based Routing

The main objective of QoS routing is to find the selection of paths that satisfy multiple QoS requirements in a network, while contributing to improved network resource utilization. QoS requirements are given as a set of constraints which can be link constraints, path constraints or tree constraints. Chen and Nahrstedt [1998] define a *link constraint* as a restriction on link usage such as link bandwidth, while a *path constraint* is an end-to-end requirement on a single path such as end-to-end delay bounds or delay jitter. A *tree constraint* specifies a QoS requirement for an entire multicast tree, for example total traffic over a network or loss ratio. Resource utilization is often computed by an abstract cost metric [Chen and Nahrstedt, 1999]. The optimization of QoS routing is then to find the lowest-cost path among all the feasible paths.

QoS-based Multicast Routing

The basics of many commonly-used multicast routing protocols such as DVMRP (Distance-Vector Multicast Routing Protocol), MOSPF (Multicast Extension to OSPF), and PIM (Protocol Independent Multicast) have been discussed in Section 2.3 of Chapter 2, Background. This section explains in more detail, how a multicast tree which is rooted at a source and spans multiple destinations is actually built.

There are several well-known multicast routing problems. The *Steiner tree* problem (also called the *least cost* multicast routing problem) is to find the tree that spans a set of destinations with the minimum total cost over all links [Hwang et al., 1992]. The *constrained Steiner tree* problem (often called the *delay-constrained least-cost* routing problem) is to find the least cost tree with bounded delay.

Multicast routing algorithms can be grouped into two categories: *source routing* and *distributed routing*. Source routing is a strategy that requires each node to maintain the global state of the network topology, and link state information. Feasible paths to each destination are computed at the source node, based on the global state information. Routing decisions are then communicated to intermediate nodes via a control message. Contrasting to source routing, in distributed (or hop-by-hop) routing each node maintains state information for nearby nodes, exchanged via control messages. Each packet contains the address of its destination, and every node maintains a forwarding table that maps each destination address into transmitting-neighbor nodes. Forwarding decisions are made by each node, upon the receipt of a packet, the destination of the packet is considered. Examples of QoS source routing algorithms are presented in [Chakraborty et al., 2003; Mokbel et al., 2000; Zhu et al., 1995]. Work in QoS hop-by-hop routing can be found in [Kompella et al., 1993; Shaikh and Shin, 1997]. These studies are described in detail in subsequent sections.

The class of QoS multicast routing problems has been shown, in general, to have high computational complexity (NP-hard) [Yuan, 2002]. Hence these algorithms usually use heuristics to reduce the complexity of the path computation problem, at the expense of not achieving an optimal solution but just a feasible solution [Paul and Raghavan, 2002]. Next, main contributions in solving constrained multicast routing problems both by source and distributed routing strategies will be discussed.

QoS-based Multicast Source Routing

In this section, some important work on source routing algorithms for multi-constrained multicasting is discussed.

Chakraborty et al. [2003] proposed an algorithm, called QoS-based Dynamic Multicast (MQ-DMR), that builds a dynamic multicast tree which satisfies multiple QoS requirements and efficiently distributes traffic throughout the network . It operates when adding a new node to the existing multicast tree. The two objectives of MQ-DMR are to minimize the overall network traffic when connecting a new receiver, and to connect a new node to the source so that QoS constraints are satisfied. In their model, a link cost is the inverse of the available bandwidth on that link. They aim to use part of the existing multicast tree to share the same path from the source to a newly added client, such that the additional traffic is minimized. The authors developed a model that dynamically assigns cost to a link based on the duration for which it would be in use. MQ-DMR operates in a similar way to the Bellman-Ford shortest path algorithm [Bellman, 1958]. In the first iteration, it tries to find one-hop least cost paths to all destinations. In the next iteration, it aims to find two-hops least cost paths with a tendency to select links that are already in use in the multicast tree. Subsequent iterations are similar. The maximum number of iteration steps can be determined based on the pre-defined allowable delay jitter.

In their experiments, Chakraborty et al. simulated their proposed MQ-DMR algorithm against Greedy [Waxman, 1986] and Naive [Doar and Leslie, 1993] multicast routing and DMG [Chakraborty et al., 1999]. DMG, which was described earlier by the same authors, is an algorithm that works similarly to MQ-DMR but is based on Dijkstra's shortest path algorithm and is not QoS-based. Greedy and Naive algorithms are dynamic multicast routing algorithms but do not take QoS requirements into account. In the Greedy algorithm, a new node joins with the shortest path to the nearest node that is in a multicast tree while in the Naive algorithm, a new node is added to a multicast tree through the shortest path from the source. Even though MQ-DMR is compared with three un-constrained algorithms, the results show that MQ-DMR performs as efficiently as the Naive, Greedy and DMG algorithms. The average tree cost results are dependent on the strictness of the QoS requirements. Regarding results, the average number of hops produced by MQ-DMR is very similar to that of the Naive algorithm.

Chakraborty et al.'s approach is quite similar to the proposed SMP, however SMP incorporates the actual size of traffic passing through a link in the cost function. SMP algorithm is based on Dijkstra's algorithm to find paths connecting a new client to an existing SMP tree through a node that is not necessarily the source.

Delay and delay variation constrained shortest path multicast routing is also a popular problem that attracts great interest from many researchers. Mokbel et al. proposed a heuristic algorithm called DVCSP that solves the problem in polynomial time and returns a near optimal tree [Mokbel et al., 2000]. The DVCSP approach includes two main phases. First, it finds a set of potential paths that satisfy the delay constraint for all destinations. This phase is facilitated by K maximum tokens kept in each node. A token at a node is used to keep track of the cost and delay experienced so far, as well as the path it has traversed from the source to the current node. Tokens are continuously duplicated from node to node until the path reaches its destination. Next, among a set of tokens arriving at a destination, one that satisfies the delay constraint variation will be chosen as the routing path reaching to that destination. The complexity of the algorithm is $O(K^2M^2)$ if $KM^2 > N^2$ and $O(KN^2)$ if $KM^2 < N^2$, where K is an integer value ranging from 1 to N, M is the number of nodes in the multicast group and N is the number of nodes in the network. Mokbel et al. compared their algorithms with other delay constrained shortest path solutions and consider two performance factors: failure rate and average cost per path. DVCSP has an additional cost to satisfy the additional constraint of delay variation.

A bounded shortest path algorithm (BSMA) to find a Delay-Constrained Least Cost (DCLC) path is proposed by Zhu et al. [1995]. Their original approach uses a loop-less k-shortest-path (KSP) algorithm [Eppstein, 1998] to list all possible shortest paths in order of increasing cost, then picks the first path that satisfies all delay requirements. The same authors proposed an alternative implementation for the BSMA heuristic bounded shortest path algorithm to reduce the time complexity of BSMA but still maintain cost performance [Feng et al., 2002]. A comprehensive comparison of different heuristics and algorithms for multicasting can be found in work of Salama and Reeves [1997].

QoS-based Multicast Hop-by-hop Routing

Though many proposed source routing algorithms to solve QoS-constrained multicast routing problems, provide promising results, they have a common lack of scalability. This weakness stems from the fact that frequent updates of global states need to be made at the source to cope with the dynamics of network and that means high computation overhead at the source [Paul and Raghavan, 2002]. To avoid this overhead, there have been attempts to solve the QoS-constrained multicast problems in a distributed manner. Two important work in this area is presented here.

Shaikh and Shin [1997] presented a destination-driven multicast routing algorithm (DDMC) that optimizes total tree cost. Most of the proposed heuristics that solve the minimum multicast tree problem assume the use of global cost information by the source, but DDMC uses only cost information from nearby nodes. The DDMC algorithm uses a greedy strategy based on Dijkstra's shortest path and Prim's minimal spanning tree algorithms. In DDMC, the costs at destination nodes are reset to zero to encourage the selection of paths that go through destination nodes. Their justification for this approach is that nodes reached from another destination node experience only an incremental additional cost, thus the total tree cost can be reduced. However this method may result in a tree with long paths connecting multiple destination nodes, so it may not meet the end-to-end delay constraint. Additionally, in their tree model, cost is not associated with any specific network parameter - however different cost metrics may have different meanings and implications on the overall performance of an algorithm. Shaikh and Shin did not provide results about the algorithm's performance on different cost metrics such as link capacity, hop distance, inverse of link bandwidth, or congestion rate; which would be beneficial to other researchers.

A distributed heuristic algorithm for solving the delay-constrained Steiner tree problem was proposed in a study by Kompella et al. [1993]. In their algorithm, every node maintains a distance vector giving the minimum delay to other nodes. The algorithm begins with only the source node in the tree and iteratively adds new links to the tree through three phases. First, the source sends a "Find message" to all nodes in the network. Upon the receipt of the "Find message", each intermediary node finds the adjacent link that can connect to a new destination (not yet included in the tree), which meets the delay constraint and minimizes a selection function. Next, the links found by all nodes are submitted to the source where the best link, the one that minimizes the selection function, is chosen. Last, an "ADD message" is sent to add the best link to the tree. The algorithm takes $O(n^3)$ time to build a tree to all destinations, where n is the number of vertices in a network.

The problem addressed in this chapter, that is finding a minimum traffic cost multicast tree for sending similar SOAP messages, can be categorized as a QoS-based routing problem. The solutions for this problem presented here are not dynamic because in every multicast session, the number of receivers are known and fixed in advance by the message aggregation process. Once the similarity of N outgoing SOAP messages is determined to be over a threshold level, a multicast tree for the client destinations of these N messages will be formed. In addition, the proposed algorithms use the source routing strategy and assume that the source has the global state information provided by a link-state routing protocol.

5.3 Notations and Problem Definition

This section formally defines the optimization problem that the tc-SMP algorithm aims to solve and outlines preliminary models and concepts before introducing the algorithms for tc-SMP routing.

The most difficult problem in tc-SMP is to find routing paths linking a source to a set of clients while simultaneously minimizing a total traffic cost function. To achieve this objective, a tree connecting the source and client nodes needs to be computed. A mathematical formulation of the *traffic-constrained similarity-based SOAP multicast* routing tree problem based on graphs can be given as follows.

Let $G(V, E)$ be a network graph, where V is the set of nodes and E is the set of edges, s is the source node, $C = \{c_1, c_2, c_3, ...c_N\}$ is the set of clients (or destinations) in the network, so $s \cup C \subseteq V$. A tree $T(s, C) \subseteq G$ originates at s and spans all members of C. Let $P_T(c_i) \subseteq T$

be the set of links in T that constitutes the path from s to client $c_i \in C$. The $tr(e)$, where $e \in E$, is a function that gives the number of bytes transmitted through link e. The total traffic generated in the tree T, denoted by *Traffic*(T), is defined as the sum of the traffic transmitted on all links in the tree, given by the expression *Traffic*$(T) = \sum_{e \in T} tr(e)$. The objective of the tc-SMP problem is to construct a tc-SMP tree $T(s, C)$ such that the tree's network traffic, *Traffic*(T), is minimized.

The traditional multicast routing tree problem (which sends *identical* messages to multiple receivers from a source while simultaneously minimizing some cost function) is an NP-hard problem [Oliveira et al., 2005]. The following lemma proves that tc-SMP is also NP-hard.

Lemma 1 *The minimum traffic-constrained similarity-based SOAP multicast routing problem (tc-SMP) is NP-complete.*

Proof This lemma is proved by method of reduction from the Steiner problem which is NP-hard [Hwang et al., 1992]. The Steiner tree problem deals with a weighted graph $G(V, E, w)$, where w is a weight function: $E \rightarrow R_+$ and a subset of vertices $D \subseteq V$. The objective of the Steiner problem is to find a minimal weight tree that includes all vertices in D [Hwang et al., 1992].

Using reduction method, one of the nodes $s \in D$ is chosen to be the source, and the rest of the nodes in D are assigned to be clients: $C = D \setminus s$. The $tr(e), e \in E$, function in the traffic-constrained SMP (tc-SMP) problem corresponds to the weight function in the Steiner problem. Therefore, the tc-SMP problem can be mapped into the Steiner problem. In addition, the weight function in the Steiner problem is static while the $tr(e)$ function in the tc-SMP problem is dynamic, depending on which message is sent across link e. Hence, the tc-SMP problem is even more complex than the Steiner problem. Thus, it can be concluded that the traffic-constrained SMP problem is NP-complete.

To cope with the NP-completeness of the problem, heuristic algorithms are developed to solve the problem in polynomial time. They are described in Section 5.4. A tc-SMP routing tree and a branching node, used in the tc-SMP algorithms, are defined as the following:

Definition 16 [*tc-SMP Tree*]: A tc-SMP tree is a tree of tc-SMP nodes. A tc-SMP r_j node is a data structure $[r_j.router, r_j.clients, r_j.cost]$ where:

1. $r_j.router$: corresponds to a physical router in the network topology,
2. $r_j.clients$: is a set of clients that r_j forwards SMP messages to, and

3. $r_j.cost$: is the total traffic already generated in a network when an SMP message arrives at r_j.

Definition 17 [*Branching Node*]: A node B is a branching node of a client C with respect to a tc-SMP tree T if the following conditions are met:

- $B \in T$.
- C is connected to the tree T through B and this connection introduces minimum additional traffic to the total tree traffic cost compared to other link connections.

5.4 Tc-SMP Routing Algorithms

Here two algorithms, *greedy* and *incremental* tc-SMP algorithms, are proposed to build a minimum traffic tc-SMP routing tree. The *greedy* algorithm is a simple approach where clients are added to a tc-SMP tree, one after another, taking the shortest path to a branching node in the tree. The *incremental* approach is however more complex, because it builds the shortest path SMP tree using the OSPF protocol first. The incremental algorithm then builds a tc-SMP tree gradually from branches of the SMP tree to ensure that the final tc-SMP tree always has less total traffic than or at least equal to that of the SMP tree. First, the two tc-SMP algorithms without using heuristics will be explained. Next, two simple heuristic methods will be proposed for use in conjunction with the tc-SMP algorithms, to improve performance.

5.4.1 Greedy tc-SMP Algorithm

The algorithm presented in this section takes a simple approach in solving the minimal traffic-constrained SMP problem. Its objective is to add a new client to the tree along a path that introduces the least additional traffic.

The multicast session begins with only the source node whose cost is intially set to zero. The greedy tc-SMP algorithm goes through multiple iterations to add paths with minimal traffic connecting each client to the tree one by one. A client may join a tc-SMP tree via a branching node (see Definition 17) where the additional traffic introduced by adding the new client is smaller than the traffic created by adding the client via other existing nodes in the tree. In order to compute the additional traffic each time a client is added, each tc-SMP node maintains two parameters (see Definition 16): (1) the list of clients that the node conveys messages to and (2) the accumulated traffic cost generated along the path from the source to the current node. In this way, the added traffic generated by a new client can be computed as the difference between the new cost and the old cost at the branching node plus the traffic generated along the shortest path from the branching node to the new client. The algorithm finishes when all clients have been successfully added to the tree.

The pseudocode of the *greedy* tc-SMP algorithm is presented in Algorithm 3 which calls another main procedure, $UpdateTreeNodeCosts$ defined in Algorithm 4. There are three main phases in each iteration adding a new client to an existing tc-SMP tree.

1. **Phase 1**: Finding the branching node (Lines 2 to 17 of Algorithm 3).
2. **Phase 2**: Updating tree nodes' costs (Line 19 of Algorithm 3, or full Algorithm 4).
3. **Phase 3**: Adding routers along the shortest path to the tree (Lines 20 to 24 of Algorithm 3).

Other functions called by the tc-SMP algorithm and the $UpdateTreeNodeCosts$ procedure are defined below.

- $ComputeCost(r, C)$ is a function that computes the accumulated cost in traffic from the source to a node r to deliver messages to a set of clients C.
- $FindSP(r, c)$ is a function to find the set of router nodes along Dijkstra's shortest path from a branching node r to a new client c.
- $SPCost(r, c)$ is a procedure that computes the amount of traffic created by sending a unicast SOAP message from a branching node r to a new client c following the shortest path.
- $Descendants(r)$ function returns a list of child nodes of an input node r in a tree.
- $SubTree(r)$ gives all the nodes in the sub-tree rooted at an input node r.
- $size(m_i)$ represents the original size of a SOAP response message m_i.
- $hops(r, r_k)$ returns the hop count between two input nodes.
- $parent(r)$ returns the lowest ancestor node of the input node r.

In addition, the following notations and assumptions are made for the proposed *greedy* tc-SMP routing algorithm:

- Let T be the set of nodes in a tc-SMP spanning tree.
- Let $M_K = \{m_1, m_2, m_3, ..., m_K\}$ be a set of SOAP response messages to be sent to all clients, and m_i be the response message for client c_i.
- Let $C_K = \{c_1, c_2, c_3, ..., c_K\}$ be a group of clients to which the SOAP messages in the M_K set are sent.
- Let *added_cost* be the additional traffic generated in the network as the result of adding client c_i into T.
- Let *min_cost* be the minimum additional network traffic as a result of adding c_i into T (the minimum of all possible *added_cost* values).

The objective of the greedy tc-SMP algorithm is to find $T = \{s, c_1, c_2, ..., c_K, r'_j, .., r'_h\}$, where r'_j is a tc-SMP node, so that *Traffic*(T) is minimized. Now, the steps involved in each phase are described in detail.

The first phase (Lines 2 to 17 of Algorithm 3) is to find the branching node through which a new client can connect to a tree such that this connection introduces the smallest additional traffic to the network compared to connections via other nodes in the tree. A random client, say c_1, is the first joining the source node. A heuristic method, which will be discussed later, can be applied to assist in choosing the first client. But, here it is assumed that there is no heuristic method applied to the greedy tc-SMP routing algorithm. The branching node for the first client is the source. The first client is always added to the tree via the shortest path based on Dijkstra's algorithm.

For subsequent clients, a branching node for a client is determined by examining through all existing tc-SMP tree nodes, (excluding leaf nodes which are clients), from top to bottom searching for potential branching nodes. A node among these potential branching nodes is then chosen to be the branching node from which the new client connects to the tree. A node is considered a potential branching node if there exists an off-tree path (none of the link resides in the tree yet) from it to the new client.

At each potential branching node, the additional traffic (*'added cost'*) introduced to the network as a result of adding the new client to the tree is computed. *'Added cost'* is calculated as the sum of the difference of the new and old costs at the branching node and the total shortest path cost from the branching node to the client. This is expressed in Line 14 of Algorithm 3. Our strategy to ensure the total traffic size is as small as possible is to minimize

Algorithm 3: tc-SMP: Minimum-traffic tc-SMP routing tree algorithm

```
1  $T \leftarrow \{s\}$
2  foreach $c_i \in C_K$ do
3      min_cost $\leftarrow \infty$
4      branch_node $\leftarrow NULL$
5      foreach $r'_j \in T$ do
6          if $|T| = 1$ then
7              branch_node $\leftarrow s$
8              break
9          else
10             old_cost $\leftarrow r'_j.cost$
11             new_cost $\leftarrow$ ComputeCost($r'_j, r'_j.clients \cup c_i$)
12             diff_cost $\leftarrow$ (new_cost $-$ old_cost)
13             sp_cost $\leftarrow$ SPCost($r'_j, c_i$)
14             added_cost $\leftarrow$ diff_cost $+$ sp_cost
15             if added_cost $<$ min_cost then
16                 min_cost $\leftarrow$ added_cost
17                 branch_node $\leftarrow r'_j$
18     $T \leftarrow T \cup c_i$
19     UpdateTreeNodeCosts(branch_node, $c_i, s$)
20     R = FindSP(branch_node, $c_i$)
21     foreach $r_k \in$ R do
22         $r'_k.clients \leftarrow c_i$
23         $r'_k.cost \leftarrow$ size($m_i$) $\times$ hops(branch_node, $r_k$) $+$ branch_node.$cost$
24         $T \leftarrow T \cup r'_k$
```

the number of bytes sent every time a new client is added to the tree. Therefore, among all potential branching nodes, the node that has the smallest *'added cost'* is selected.

The second phase (Line 19 of Algorithm 3, or Algorithm 4) is to update the costs of all existing nodes in the tree to reflect the addition of the new client. This pseudocode for this phase is given in Algorithm 4 (*UpdateTreeNodeCosts*). It updates the *clients* lists, denoted as *r.clients*, and calculates the accumulated *costs*, *r.cost*, of a branching node and its ancestors (Lines 6 and 7 in Algorithm 4). A *clients* list is a list of clients that a node forwards SMP messages to and the accumulated *cost* of a node is the total traffic generated over the network when an SMP message arrives at the node as defined in Definition 16. The costs of nodes that are in the sub-trees of the ancestor nodes of the branching node are also incremented

Algorithm 4: *UpdateTreeNodeCosts* Procedure: Update the costs of all nodes in a tree

```
Input: s: The source
branch_node: Branching node
c: The newly added client
1  procedure UpdateTreeNodeCosts do
2      previous_root ← $NULL$
3      $r$ ← branch_node
4      while $r = s$ || parent($r$) $\in T$ do
5          old_cost ← $r.cost$
6          $r.clients$ ← $r.clients \cup$ c
7          $r.cost$ ← ComputeCost($r, r.clients$)
8          diff_cost ← ($r.cost$ − old_cost)
9          foreach $u \in$ Descendants($r$) do
10             if $u \neq$ previous_root then
11                 foreach $v \in$ SubTree($u$) do
12                     $v.cost$ ← $v.cost$ + diff_cost
13         if $r \neq s$ then
14             previous_root ← $r$
15             $r$ ← parent($r$)
16         else quit
```

by a *diff_cost* amount (Lines 9–12). *Diff_cost* is computed as the difference between the cost of the ancestor node before the update and the new updated cost.

The third phase (Lines 20 to 24 of Algorithm 3) is to add the nodes, that are along the shortest path from the branching node to the new client, to the tree. The *clients* list, $r'_k.clients$, of a node r'_k among these newly added routers will include only the new client. The accumulated cost of r'_k, $r'_k.cost$, is the sum of the branching node's accumulated cost and the multiplication of h and the size of a unicast SOAP message, where h is the number of hops from r'_k to the branching node.

Example Illustration

The diagram in Figure 5.1 is an example to illustrate the greedy algorithm. If a router r_i from the topology is included in the spanning tree T, it corresponds to a node r'_j in the tree T where $r'_j.router = r_i$. At the beginning, the tree includes only the source node: $T = \{r'_0\}$, and c_1 is the first client to be added to T. The shortest path from the source s to the client

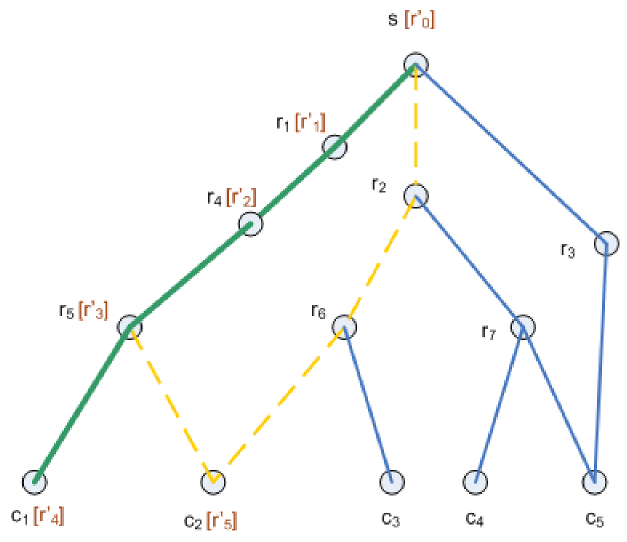


Figure 5.1: A sample network used to illusatrate tc-SMP routing

c_1 (highlighted in the bold line) is then added to the tree. The spanning tree T is then updated to include $\{r'_0, r'_1, r'_2, r'_3, r'_4\}$. The cost at node r'_1 is the amount of traffic created by sending a message addressed to c_1 over link l_{s-r_1}. The cost at node r'_2 is the total size of traffic generated by sending a message to c_1 over links l_{s-r_1} and $l_{r_1-r_4}$.

The next client to be added to the tree is c_2. The routing path for c_2 will be chosen in consideration of the path already determined for c_1, so that the total traffic volume created as the result of sending messages to these two clients is minimal. From the network topology in Figure 5.1, there are two feasible paths to client c_2 from the source s via different forwarding routers r_1 and r_2. The source node must decide on path that will create the smallest amount of traffic together with the message for client c_1.

The dotted lines represent paths from two potential branching nodes, s and r_5 in the network topology, for connecting client c_2 to the existing tc-SMP tree T. Assuming that messages to c_1 and c_2 are exactly the same, r_5 $[r'_3]$ will be chosen as the branching node because it has the smallest additional traffic cost when including c_2 in the client list. Node c_2 [r'_5] will be added as a new node in T as well.

Finally, all routers that are on the newly found optimal path from the branching node to the new client will also be added to the spanning tree. The costs of all nodes in T will be updated to reflect the new status of the tree in terms of traffic (with the inclusion of the new client). Repeating this process for the rest of the clients. Adding clients one by one to the spanning tree will result in a full spanning tree to all destinations in the group. The complexity of the *greedy* algorithm is analysed in Section 5.4.5.

5.4.2 Incremental tc-SMP Algorithm

The second approach to solve the tc-SMP problem is called *incremental* tc-SMP. While the greedy algorithm builds a tc-SMP tree from scratch, the incremental algorithm builds a tc-SMP tree based on a shortest path SMP tree. The incremental tc-SMP algorithm iteratively examines each branch in the SMP tree connecting to a client and determines if the branch can be replaced by an alternative path in the network to reduce the traffic created when transmitting an SMP message to that client. If there is no such a substitute branch, the shortest path to the client from the SMP tree will be added to the tc-SMP tree. Therefore, in the worst scenario, incremental tc-SMP produces the same results as SMP.

Algorithm 5 formally shows the steps involved in this algorithm. The steps are grouped into three main phases as presented in the following list. The details of each phase are elaborated in the subsequent paragraphs.

1. **Phase 1**: Setting up an SMP tree based on shortest paths (Line 1 of Algorithm 5).
2. **Phase 2**: Finding alternative paths connecting to each client (Lines 15-16 of Algorithm 5).
3. **Phase 3**: Building a temporary tree, T_{temp}, which includes the newly found alternative path to the client (Lines 18-19 of Algorithm 5 and Algorithm 6).
4. **Phase 4**: Selecting the branch that connects to the client with the least traffic to add to the tc-SMP tree (Lines 20-23 of Algorithm 5).

The following notations and assumptions are made for the proposed *incremental improvement* tc-SMP routing algorithm:

- Let T_{tc-SMP} be the set of routers in the tc-SMP spanning tree.
- Let T_{temp} be a temporary tree built during the path finding process for each client.
- Let T_i be a tree generated after each iteration of adding a new client to the tc-SMP spanning tree.
- Let $C_K = \{c_1, c_2, c_3, ..., c_K\}$ be a group of clients that have requested for $|M_K|$ SOAP messages which have a similarity level greater than or equal to the required threshold and can be aggregated into one SMP message.

Algorithm 5: Incremental tc-SMP algorithm

```
$T_{SMP} = \{r_k, r_{k+1}, ..\}$: an SMP tree
$T_{temp} \leftarrow T_{SMP}$; $T_{tc-SMP} \leftarrow \emptyset$
foreach $c_i \in C_K$ do
    if $T_{tc-SMP} = \emptyset$ then
        foreach $r_j \in p_{s,c_i}$ do
            $\{p_{s,c_i}$ is the shortest path from $s$ to $c_i\}$
            $T_{tc-SMP} \leftarrow T_{tc-SMP} \cup r_j$
        $T_i \leftarrow T_{SMP}$
        continue
    else
        $\{p^{min}_{s,c_i}$ is a path from $s$ to $c_i$ that introduces the least traffic in the tree$\}$
        $p^{min}_{s,c_i} \leftarrow p_{s,c_i}$ , where $p_{s,c_i} \in T_{i-1}$
        $T_i \leftarrow T_{i-1}$
        foreach $r_j \in T_{tc-SMP}$ do
            if $r_j \neq s$ && $r_j \notin C_k$ && $r_j \notin p_{s,c_i}$ then
                $\mathsf{R} = \mathtt{FindSP}(r_j, c_i)$
                if $\mathsf{R} \neq \emptyset$ then
                    $T_{temp} \leftarrow \mathtt{RemoveClientFromTree}(T_{temp}, c_i)$
                    $T_{temp} \leftarrow T_{temp} \cup \mathsf{R} \cup p_{s,r_j}$
                    if $\mathtt{Cost}(T_{temp}) < \mathtt{Cost}(T_i)$ then
                        $p^{min}_{s,c_i} \leftarrow p_{s,r_j} \cup p_{r_j,c_i}$
                        $T_i \leftarrow T_{temp}$
        $T_{tc-SMP} \leftarrow T_{tc-SMP} \cup p^{min}_{s,c_i}$
```

- Let $M_K = \{m_1, m_2, m_3, ..., m_K\}$ be the set of SOAP response messages to be sent to all clients; m_i is the response message for client c_i.
- Let $Cost(T)$ be a function of the total traffic cost of the whole tree T.
- Let $FindSP(r_j, c_i)$ be a function to find a list of routers connecting router r_j to client c_i along its shortest path.

In the first phase, the source establishes a routing tree, called T_{SMP}, where every path from the source to each client is based on Dijkstra's shortest path first algorithm, similar to the original SMP algorithm. This phase can be easily completed by using the OSPF

Algorithm 6: *RemoveClientFromTree* Procedure: Remove nodes spanning a client from a tree

```
Input: $T$: the input tree
c: The client to be removed from $T$
Output: $T$: The resulted tree after removing c
1 procedure RemoveClientFromTree do
2     foreach $r_i \in p_{s,c}$ do
3         if $|r_i.clients| = 1$ then
4             {There is only c passing through this router }
5             $T \leftarrow T \setminus r_i$
```

routing protocol deployed in most networks. Each node r_j in the T_{SMP} tree aslo maintains the properties $r_j.router$, $r_j.clients$ and $r_j.cost$ as defined in Definition 16.

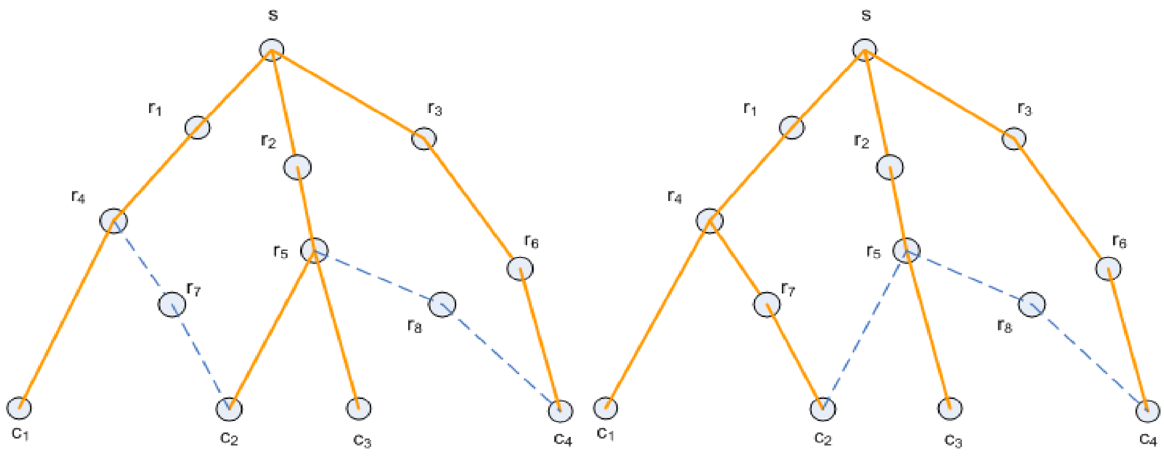
The tc-SMP tree, T_{tc-SMP}, is initially empty. Each client is added to the tc-SMP tree, one after another. In the first iteration, from K clients, a random client c_1 is chosen as the first client to be added to the T_{tc-SMP} tree. The first client is a special case where the tc-SMP path to c_1 is the same as the SMP path to c_1 (see Lines 4–7 of Algorithm 5). This is similar to the greedy approach in that the first client is always added to the tree using the shortest path first algorithm. The current total cost in terms of traffic in the tree will be calculated and denoted by $Cost(T_1)$.

In the next iteration, the algorithm attempts to finding an alternative path for another client, say c_i, as illustrated from Lines 11–23 of Algorithm 5. T_{temp} is denoted as a temporary tree which is built during the process of finding an alternative path for a client c_i. The tc-SMP algorithm examines if there exists a path from a node (excluding the source and the clients), r_j, that is already in T_{tc-SMP}, to the client c_i. Also, the algorithm ignores r_j if it resides on the original shortest path p_{s,c_i} from the previous tree.

If there is a path from r_j to c_i that meets the criteria in Line 15 of Algorithm 5, the T_{temp} is then built based on T_{i-1}, which is the resulted tree from the previous iteration, but excluding the path spanning c_i (see Line 18 of Algorithm 5 and full Algorithm 6). Subsequently, the path p_{r_j,c_i} is added to T_{temp}. The total cost of T_{temp} is then calculated as a result of this and compared to the cost of T_{i-1}. If $Cost(T_{i-1})$ is greater than $Cost(T_{temp})$, T_{temp} will be assigned to T_i. This process continues until all the clients in the list are examined for alternative paths.

Example Illustration

Figures 5.2 and 5.3 show an example of how the incremental tc-SMP algorithm operates. In this example, there are 4 clients c_1, c_2, c_3 and c_4 all requesting the same SOAP message. Let ω denote the size of each SOAP response message. To make the example easy to follow, the size of an aggregated SMP messages of any number of standard SOAP messages is still ω. First, an SMP tree, denoted by T_{SMP}, is built by using the OSPF protocol. T_{SMP} roots at the source s and spans all four clients on the shortest paths, as shown in solid lines in Figure 5.2(a). The cost in terms of the total traffic created if SMP messages are sent following the paths in T_{SMP} is then computed ($Cost(T_{SMP}) = 10\omega$). Next, a tc-SMP tree, denoted by T_{tc-SMP}, will be built gradually by adding the clients to the tree one after another. The first client, c_1, is added to the tc-SMP tree through the shortest path, therefore, the source s, routers r_1 and r_4 and client c_1 are included in the T_{tc-SMP} tree.



(a) Initial SMP tree (in solid lines)

(b) A temporary tree (in solid lines)

Figure 5.2: An SMP tree and a temporary tree built by the incremental tc-SMP algorithm in a sample network

To add client c_2 to T_{tc-SMP}, the algorithm examines if there exists a path from a node, that resides in T_{tc-SMP} and satisfies the criteria on Line 15 of Algorithm 5, to c_2. Routers r_1 and r_4 both have paths to c_2. Let us consider r_4 as the branching node for linking c_2 as r_4 is closer to c_2 than r_1 is. A temporary tree, denoted by T_{temp}, is built by removing the single path spanning c_2 in the T_{SMP} tree and adding the new path connecting r_4 to c_2. This T_{temp} tree is depicted in Figure 5.2(b). The cost of this temporary tree is computed and equal to 11ω which is higher than the cost of the initial T_{SMP} tree (10ω). Therefore, in this iteration the shortest path from the source to client c_2 is added to T_{tc-SMP}. If the greedy tc-SMP algorithm was used instead, r_4 would be chosen as the branching node over the source because the greedy algorithm only considers the total traffic cost for clients that it has added so far instead of the whole client set.

Client c_3 can be added to T_{tc-SMP} easily because there is only one path from the source to c_3. There are two paths spanning client c_4, and it is trivial to realise that c_4 will be added to T_{tc-SMP} via r_5. The final incremental tc-SMP tree is illustrated in Figure 5.3 along lines with arrows.

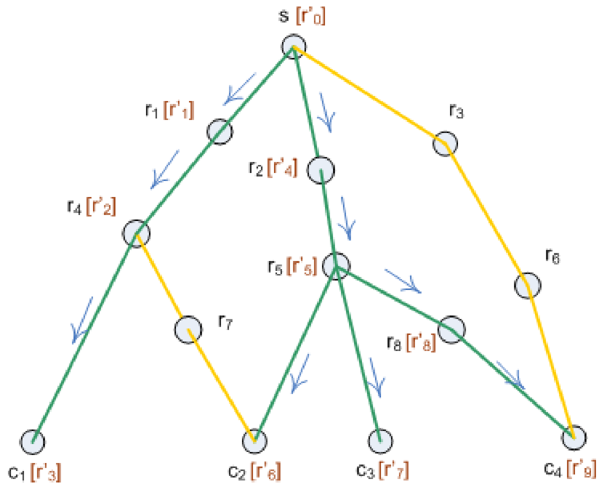


Figure 5.3: A tc-SMP tree (lines with arrows) finally built by the incremental tc-SMP algorithm in a sample network

5.4.3 Heuristic Methods

The basic functionalities of the two tc-SMP routing algorithms without using a heuristic have been described. Here, two simple methods, which can be applied to both algorithms to determine the order of clients in a distribution list to be added to a tree, are proposed.

Message size-based Heuristic Approach

This approach is based on the sizes of response messages. First the K SOAP response messages are sorted in a descending order according to their size. Clients are added to the tree in order by their descending message size. Using this method, large messages are sent out first along the least hop paths. Thus less additional traffic is generated later.

Similarity-based Heuristic Approach

This heuristic method is based on the similarities between response messages. The first client to be added to a tree is the one that has the largest message size. Then subsequent clients are added to the tree in order by descending message similarity with existing clients' messages in the tree. With this method, messages with higher similarity tend to be sent along paths that have more common links, thus more network bandwidth can be saved.

5.4.4 Algorithm Analysis

For both proposed algorithms, a number of tc-SMP trees may need to be built to deliver all the responses to the clients. This is because a tc-SMP multicast tree is only for a group of clients whose SOAP response messages can be combined. Information about the tree will be piggied-back on SMP messages so that intermediary routers can look up the routing information and determine next-hop routers for each client embedded in the messages.

The main difference between the *greedy* and *incremental* algorithms is that the *greedy* one builds a tc-SMP tree from scratch while the *incremental* one builds a tc-SMP tree based on an established SMP tree and adds paths that minimize the total cost of the tc-SMP tree. The *incremental* algorithm guarantees that every time a path for a client is chosen, a tc-SMP tree produces an equal to or smaller traffic volume than an SMP tree.

5.4.5 Complexity Analysis

The time complexity of the proposed algorithms to build a tc-SMP routing tree is analyzed here. The computation time for the algorithms is shown to be polynomial by proving the following lemmas.

Lemma 2 *The time complexity of the greedy algorithm to build a traffic-constrained SMP routing tree is $O(n(m + nlogn))$, where m is the number of edges and n is the number of nodes in a network.*

Proof In a network graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. In the worst case, all clients in the set C will be added to a tc-SMP tree. The *for-loop* from Lines 2 to 24 of Algorithm 3 is executed once for each client and hence for a total of N times. The *for-loop* of Lines 5–17 of the same algorithm, in the worst case, is performed n times if all the nodes in the set V are already included in the routing tree. In each round of this loop, an execution of finding the shortest path between the current node to a client takes place. Dijkstra's shortest path algorithm is run in $O(m + nlogn)$ time using a Fibonacci heap implementation [Cormen et al., 2001]. Therefore, the complexity to complete the full *for-loop* of Lines 5–17 is therefore $O(n(m + nlogn))$. The $UpdateTreeNodeCosts$ procedure in the worst case needs to update all the nodes in the network, thus has a time complexity of $O(n)$. Hence, total execution time of the entire algorithm is $O(N(n(m + nlogn) + n))$, which can be simplified to $O(n(m + nlogn))$ since N can be considered as a constant as there is always a limit on the number of messages aggregated in an SMP message. In conclusion, the *greedy* tc-SMP algorithm runs in polynomial $O(n(m + nlogn))$ time.

Lemma 3 *The time complexity of the incremental algorithm to build a traffic-constrained SMP routing tree is $O(n(m + nlogn))$ where m is the number of edges and n is the number of nodes in a network.*

Proof The worst case for this algorithm is when building a tc-SMP tree that spans all N clients. Considering the main for-loop, Lines 3–23 of Algorithm 5. This *for-loop* is executed once for each client and hence for a total of N times. As explained above, N is considered as a constant in this analysis as there is often an upper bound on the number of clients that can be aggregated in an outgoing SMP message from the server. Inside this loop, the algorithm runs through all the nodes that already exist in the tc-SMP tree, which in the worst case would have n nodes. Finding the shortest path from a node to a client (as presented in Line 16 of Algorithm 5) requires $O(m + nlogn)$ time using a Fibonacci heap implementation [Cormen et al., 2001]. Line 18 calls the *RemoveClientFromTree* procedure, described in Algorithm 6, which requires $O(L)$ time where L is the largest number of hops from the source to any client in the network. The time complexity required to measure the cost of a temporary tree, Line 20, is $O(n)$. Therefore, the *for-loop* of Lines 3–23 takes $O(n(m + nlogn) + L + n)$, simplified to $O(n(m+nlogn))$, to complete. In conclusion, execution time of the *incremental* tc-SMP algorithm is of the order $O(n(m + nlogn))$ time.

The proposed algorithm is more complex than the OSPF algorithm, because the OSPF problem finds the optimal path for each destination independently. In contrast, the traffic-constrained SMP problem involves path optimizations for multiple destinations the paths of which are dependent on each other and there is a common tree constraint of reducing the overall network traffic.

Symbol	**Description**
N	Number of clients.
K	Number of operations.
opt_i	One in K operations offered by a Web service provider.
N_i	Number of clients accessing opt_i.
s	The source node.
C	The set of clients.
$\mathit{Traffic}\{A\}$	The total network traffic when using routing protocol A.
$\mathit{Diff\_Traffic}\{A, B\}$	The difference in total traffic between routing protocols A and B.
$size_{msg}$	The average size of a SOAP message.
h	Number of hops along the shortest paths from a source to all clients except c_1, $\{C \setminus c_1\}$, in the best case total traffic scenario.
$\mathit{Diff\_response}\{A, B\}$	The difference in average response time between A and B.
$t_{procServer}(A)$	The server processing overhead when using routing protocol A.
$t_{procSim}$	The processing overhead to measure the similarity between messages (common for both SMP and tc-SMP).
$t_{treeBuilder}$	The processing overhead to build a routing tree under tc-SMP.
t_{path}	The time it takes to find the shortest path from a router to a client and to measure the additional cost added to a tree if the client is added through that router.
n	Number of nodes in a network graph.

Table 5.1: Legend of symbols used in tc-SMP theoretical models

5.5 Analytical Study

In this section, the analytical models of the total network traffic and the average response time for the tc-SMP routing protocol are presented and compared with the analytical models of SMP, multicast and unicast protocols, which were presented in the previous chapter. These models will enable a comparison to be made between various SOAP multicast schemes without the need for simulation. This analysis will also provide a general framework for analyzing the performance of these schemes under different workloads and network sizes and will be used to verify the accuracy of simulation results.

Table 5.1 outlines the definitions of symbols and notations used in the analytical study. In addition, the network and Web service operation models are defined as per in Chapter 4.

5.5.1 Total Network Traffic

Computing the performance metrics (traffic and response time) of tc-SMP theoretically is a complex task. To compare tc-SMP against SMP, it is necessary to model a tc-SMP routing tree based on the same network which was used to model SMP. It is relatively straightforward to model an SMP routing tree in a network where multiple paths exist from one node to another. In such a tree, the source as the root is connected to its clients through the shortest paths. The shortest paths can be easily found using Dijkstra's algorithm, as explained in Chapter 4. However, it is difficult to estimate the average response time and total bandwidth of a tc-SMP tree due to the influence of many parameters (such as the layout of the network, and the similarity level between SOAP messages).

The complexity of a network topology presents a challenge in computing the total network traffic. Assuming a fixed network topology such as a binary tree network, is a poor option because it does not provide a generalization of the problem. In this chapter, theoretical models for tc-SMP are based on the models previously used for SMP in Chapter 4, Section 4.7. The difference in total network traffic between the tc-SMP and SMP routing schemes is then determined. Specifically, worst and best case scenarios are approximated to find the range of difference in total network traffic between the two.

A) Worst case

The worst case, analysed in this section, refers to the case where the difference in traffic, $\textit{Diff\_Traffic}\{SMP, \text{tc-SMP}\} = \textit{Total\_Traffic}\{SMP\} - \textit{Total\_Traffic}\{\text{tc-SMP}\}$, is smallest. The worst case scenarios for both greedy and incremental algorithms are examined.

Greedy tc-SMP Algorithm: The scenario where the *greedy* tc-SMP algorithm performs worse than SMP is when the shortest-path tree built by SMP is already an optimal tree, and the tc-SMP tree built using the greedy algorithm is only a near optimal one. Since, under the greedy approach, a source does not have a full view of routing paths to all clients, when adding a new client to a tree, the greedy algorithm can only optimize the traffic for clients that already exist in the tree, not all clients. Therefore, it is not expected that traffic in the greedy tc-SMP tree is always smaller than that in the SMP tree. The worst difference in traffic for the greedy algorithm is given below:

$$\textit{Total\_Traffic}\{SMP\} \leq \textit{Total\_Traffic}_{worstcase}\{\text{tc-SMP}(greedy)\} \tag{5.1}$$

The above inequality can be re-written as follows:

$$\textit{Diff\_Traffic}_{worstcase}\{SMP, \text{tc-SMP}(greedy)\} \leq 0 \tag{5.2}$$

Incremental tc-SMP Algorithm: Since the *incremental* tc-SMP algorithm builds a tc-SMP tree based on an SMP tree, in the worst case, the traffic of the tc-SMP tree is equal to that of the SMP tree. Therefore, the following inequality can be drawn for the difference in total traffic between the *incremental* tc-SMP algorithm and SMP given the same number of clients and network topology.

$$\textit{Diff\_Traffic}\{\textit{SMP},\ \textit{tc-SMP(incremental)}\} \geq 0 \tag{5.3}$$

B) Best case

The best-case scenario here implies the case when the difference in traffic between SMP and tc-SMP, denoted as *Diff_Traffic*$\{SMP, \text{tc-SMP}\}$, is largest. This scenario occurs when all messages to N clients can be sent as an aggregated message along h hops from a source to router r_h, where the SMP message is split into N individual unicast messages to all the clients. Figure 5.4 depicts this scenario, the solid lines represent the tc-SMP paths. There are $h+1$ hops from the source to any of the clients using tc-SMP routing. SMP routing

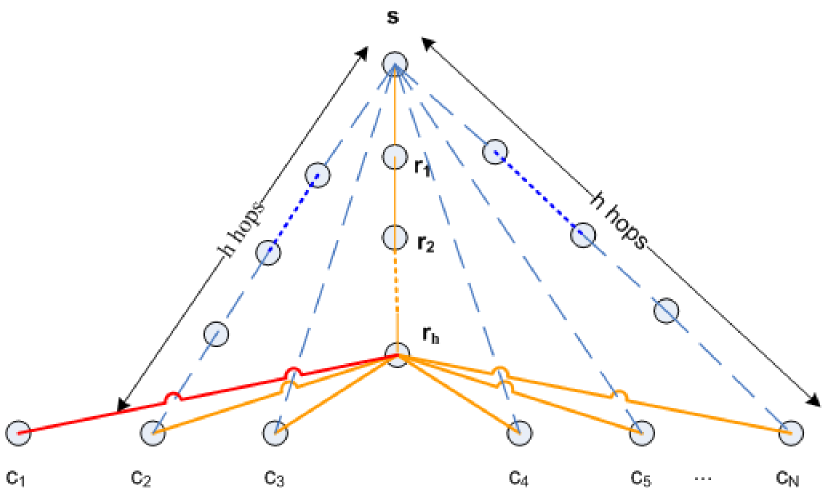


Figure 5.4: Best total traffic scenario with tc-SMP routing

paths are represented by dashed lines. The SMP routing tree has only one common path, which is p_{s,c_1} connecting the source to the first client c_1, with the tc-SMP routing tree. Other paths in the SMP tree are h-hop shortest paths from the source to all clients in $\{C \setminus c_1\}$. For simplicity, it is assumed that the size of each original SOAP response message is all the same and is equal to $size_{msg}$. The total traffic generated by the SMP paths is given by the following equation:

$$\textit{Total\_Traffic}\{\textit{SMP}\} = (h+1)size_{msg} + h(N-1)size_{msg} = (1 + h \times N)size_{msg} \tag{5.4}$$

The total traffic generated on the tc-SMP paths is equal to the sum of the traffic along the common path from s to r_h and the traffic at the final hops from r_h to each of the clients. Along the common path, the size of the aggregated message is assumed to be $\alpha \times size_{msg}$, where $1 \leq \alpha < N$ (the aggregated message size is always equal to or greater than the original response message size and less than the sum of all original SOAP messages). The total traffic for tc-SMP is given by the following equation:

$$\textit{Total\_Traffic}\{\textit{tc-SMP}\} = h \times \alpha \times size_{msg} + N \times size_{msg}$$

In the best-case scenario, the similarity among all messages is 1, so $\alpha = 1$, and the above formula becomes:

$$\textit{Total\_Traffic}\{\textit{tc-SMP}\} = h \times size_{msg} + N \times size_{msg} = (h+N) \times size_{msg} \tag{5.5}$$

From Equation 5.4 and Equation 5.5 the best case traffic difference between tc-SMP (both greedy and incremental algorithms) and SMP is given by:

$$\textit{Diff\_Traffic}\{\textit{SMP,tc-SMP}\} \leq (h \times N + 1 - h - N) \times size_{msg} \tag{5.6}$$

In conclusion, from Equations 5.2, 5.3, and 5.6, the ranges of differences in total network traffic generated between SMP and greedy tc-SMP algorithms; and between SMP and incremental tc-SMP are shown below:

$$\begin{aligned} a \quad &\leq \quad \textit{Diff\_Traffic}\{\textit{SMP, tc-SMP(greedy)}\} \leq (h \times N + 1 - h - N) \times size_{msg} && (5.7)\\ & \quad\ \ \textit{where} \ \ a \ \ \leq 0 \\ 0 \quad &\leq \quad \textit{Diff\_Traffic}\{\textit{SMP, tc-SMP(incremental)}\} \leq (h \times N + 1 - h - N) \times size_{msg} && (5.8)\end{aligned}$$

Expression 5.7 implies that greedy tc-SMP generally outperforms SMP by a certain amount of traffic up to $(hN+1-h-N)size_{msg}$ units of bytes, but in some cases, greedy tc-SMP can be outweighed by SMP. On the other hand, Expression 5.8 shows that incremental tc-SMP always produces less traffic than or equal to SMP.

5.5.2 Average Response Time

Modeling a function to accurately compute average response time is a complex task, since the result is influenced by the topology of a network. The difference in average response time between SMP[2] and tc-SMP is mainly due to the overhead at the server for building tc-SMP routing trees. For simplicity, it is assumed in this model that the total transmission delay, propagation delay and processing delay at intermediate nodes experienced by a client are the same in both tc-SMP and SMP. Therefore, the difference in average response time between SMP and tc-SMP is also the difference in the server processing overhead time between them. This is shown in the following expression:

$$\textit{Diff\_response}\{\textit{SMP,tc-SMP}\} = t_{procServer}(SMP) - t_{procServer}(\text{tc-SMP}) \tag{5.9}$$

Both SMP and tc-SMP have the same overhead in measuring message similarity and in determining which messages can be aggregated. This time is denoted by $t_{procSim}$, which is the same as $t_{procServer}(SMP)$ defined in Section 4.7.3. Therefore, the following equation is obtained:

$$t_{procSim} = t_{procServer}(SMP) \tag{5.10}$$

Under tc-SMP, extra time overhead, denoted by $t_{treeBuilder}$, is required to build a tc-SMP routing tree. As proved previously, both greedy and incremental tc-SMP algorithms have the same time complexity. Therefore, the server processing overhead experienced by a client for both tc-SMP algorithms can be expressed as follows:

$$t_{procServer}(\text{tc-SMP}) = t_{procSim} + t_{treeBuilder} \tag{5.11}$$

The process of finding the branching node for a new client in tc-SMP involves searching each existing node in a T_{tc-SMP} tree to find the shortest path from the node to the client, and computing the traffic cost introduced by adding this shortest path to the tree. The time it takes to do this on each router is called t_{path}. The maximum number of nodes to be processed when a client is added to the tree is n, which is the number of vertices in a network graph. Therefore, in the worst case, the processing overhead to build a tc-SMP tree for N clients

[2] Under SMP, messages are routed automatically using OSPF, a common routing protocol employed by most networks.

can be given as:

$$t_{treeBuilder} = N \times n \times t_{path} \tag{5.12}$$

Combining Equations 5.9, 5.10, 5.11 and 5.12, the average response time difference between SMP and tc-SMP is given as below:

$$\textit{Diff\_response}\{\textit{SMP,tc-SMP}\} = t_{procServer}(SMP) - t_{procSim} - N \times n \times t_{path} \tag{5.13}$$

As explained in Equation 5.10 $t_{procServer}(SMP) = t_{procSim}$, Equation 5.13 can be rewritten as:

$$\textit{Diff\_response}\{\textit{SMP,tc-SMP}\} = -N \times n \times t_{path} \tag{5.14}$$

Equation 5.14 shows that tc-SMP experiences slower average response time than SMP by $N \times n \times t_{path}$ time units.

5.6 Simulation and Results

The purpose of this section is to identify the characteristics of the proposed routing algorithm and evaluate its performance under various network conditions. To test the effectiveness of tc-SMP over SMP, its performance is evaluated and compared with SMP, traditional multicast and unicast communications. Details of the experimental setup and simulation results are described in this section.

5.6.1 Experimental Setup

The OMNeT++ [Varga, 2006] simulation program was used to randomly generate different hierarchical network topologies to carry out experiments. Figure 5.5 illustrates a sample network of 40 clients used for testing. In these topologies, the maximum number of hops in the shortest paths from the source to any client is 10. The propagation delay, which is the time that a network message takes to travel from one node to another is constant at $t_{prop} = 5ms$. The topologies were generated such that there were always multiple paths to convey a message from the source to most of the clients. The number of clients in the network ranged from 10 to 200. For each network topology, six tests were performed. They were greedy tc-SMP without heuristic, incremental tc-SMP without heuristic , greedy tc-SMP with the message size-based heuristic, incremental tc-SMP with the message size-based heuristic, greedy tc-SMP with the similarity-based heuristic and incremental tc-SMP with the similarity-based heuristic. The test results for SMP, traditional multicast and unicast were obtained from the experience described in the previous chapter. For each test, 20 experimental runs were performed and the result given is the average of these runs.

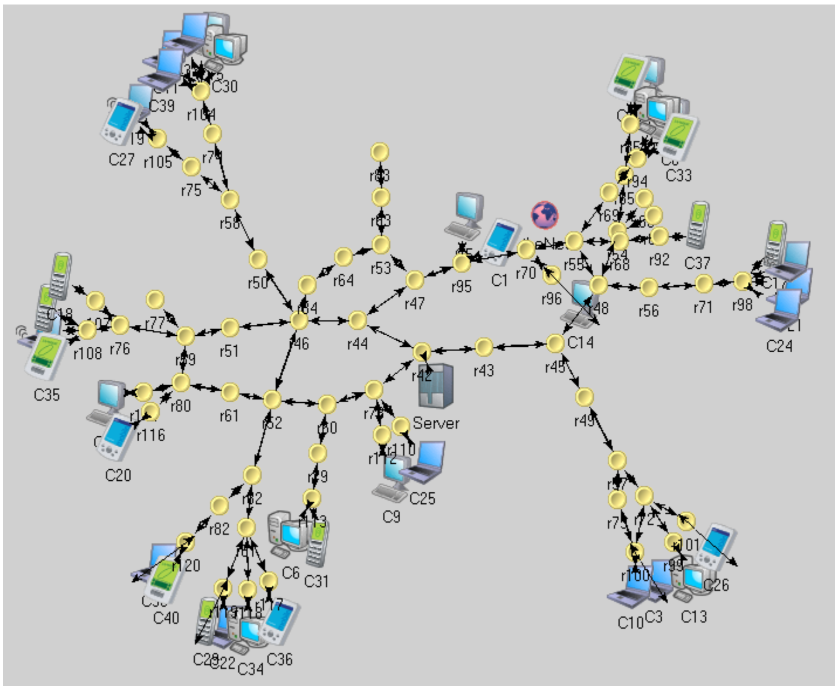


Figure 5.5: Sample of a simulated network topology

Client requests to the Web service operations followed a Zipf distribution [Zipf, 1949] with a skewness factor of $\alpha = 1$. There were 10 operations defined in the Web service's description document in the experiments. These operations correspond to 10 SOAP response messages denoted by $m_1, m_2, \ldots, m_{10}$ in which m_1 is the most frequently requested message and m_i is the i^{th} frequently accessed one. The size of the messages ranged from 20Kb to 50Kb. The similarity threshold used for SMP and tc-SMP methods is 0.7. The similarity between messages depends on the requested service operation and on its input parameters. The simulated bandwidth available on each link was 1.5Mbps.

5.6.2 Experimental Results

The total network traffic and the average response time for each client are the two metrics used to examine the performance of the tc-SMP algorithm and to compare with SMP, multicast and unicast. The network load is the total size of all messages passing through all links in the routing tree when sending responses to all the clients. The average response time is the average time it takes from the server sending a response message out, until the message reaches the destined client. It is computed by dividing the sum of the delays that each client experiences by the number of clients. The response time includes propagation and transmission delays on each link and processing delays at the server and at intermediary nodes.

A) General Case

This section presents the results for experiments in the general cases as outlined in Section 5.6.1.

Total Network Traffic: Figure 5.6 shows the total network traffic for greedy and incremental tc-SMP algorithms compared to SMP, multicast and unicast schemes. As expected, unicast produces the greatest volume of traffic, that is proportional to the number of receivers. Traditional multicast protocol represents an improvement of around 30% over unicast, while SMP and tc-SMP can reduce traffic by up to 50% and 65% respectively. With a small network of under 50 clients, the reductions in traffic between tc-SMP and SMP over unicast are quite small, with little difference between them (around 15%).

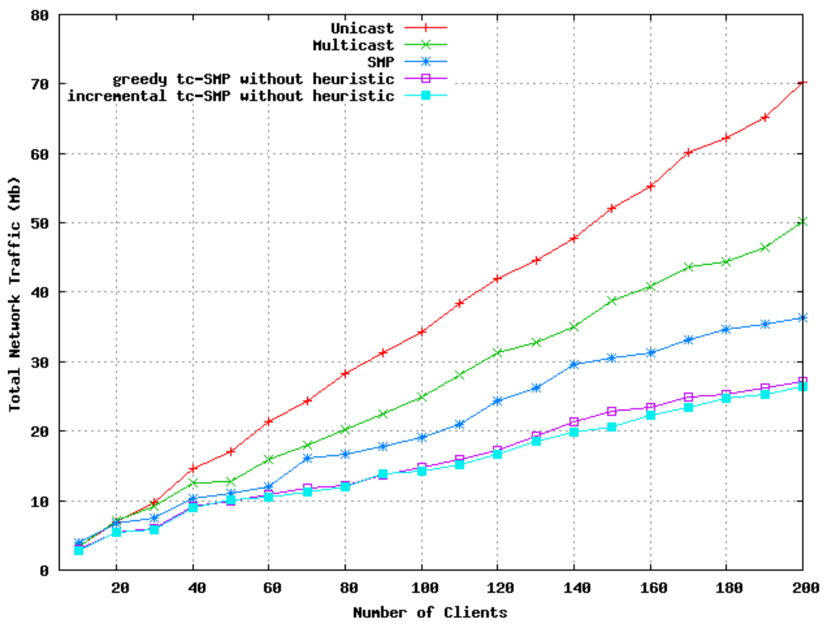


Figure 5.6: Total network traffic with different routing protocols.

With larger networks (100 to 200 clients) tc-SMP's and SMP's performance gain over unicast in traffic is more significant — over 60% for tc-SMP and over 45% for SMP. Comparing tc-SMP to SMP, the difference in bandwidth consumption is not noticeable with small networks of 10 or 20 clients. When the client numbers increase to 50, 100 and 200, tc-SMP (both greedy and incremental algorithms) outperforms SMP by around 10%, 20%, and 25% respectively. As shown from Figure 5.6, the *incremental* tc-SMP algorithm in general performs better than the *greedy* tc-SMP algorithm by 2–4% across all network sizes.

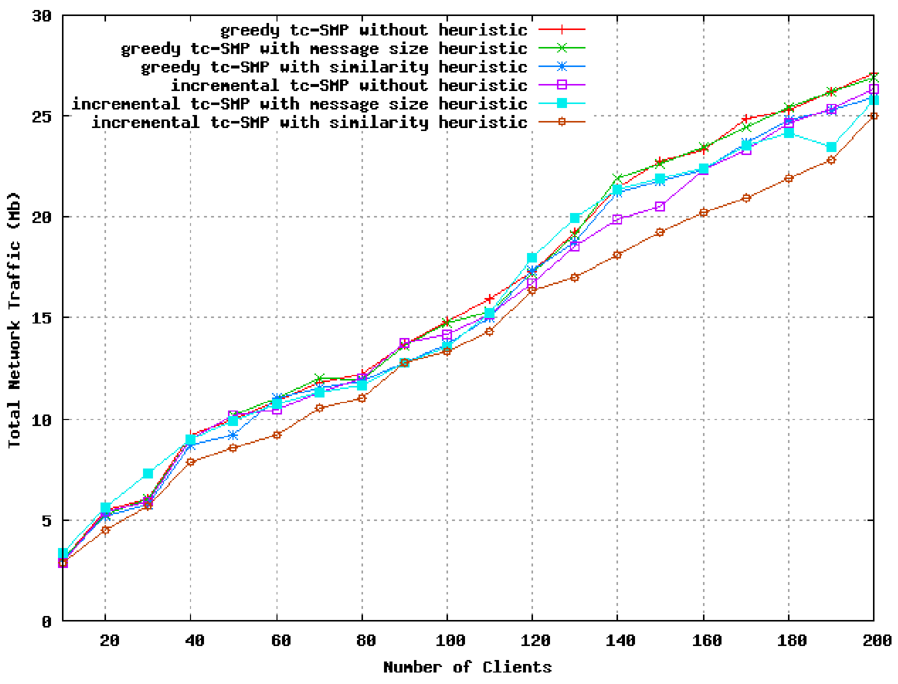


Figure 5.7: Total network traffic with different heuristics and non-heuristic tc-SMP algorithms

Figure 5.7 compares the total network traffic for different tc-SMP algorithms with and without heuristics. In general, there is no significant difference between the network traffic results with or without tc-SMP heuristics. A close look reveals that the similarity-based heuristic method presents an improvement of approximately 3% over its message size-based heuristic method counterpart. For example, with a network of 100 clients, the *greedy* tc-SMP with message-size based heuristic produces 14.74Mb traffic while the similarity-based heuristic for the same tc-SMP algorithm produces only 13.68Mb traffic. Similarly, with a network of 150 clients, the *incremental* tc-SMP algorithm with message-size based heuristic and similarity based heuristic generate 21.9Mb and 19.23Mb traffic respectively. It is evident that the *incremental* tc-SMP algorithm with the similarity based heuristic method produces least traffic of the tc-SMP variations by a margin of around 5%.

Average Response Time: The average response times observed in the experiment is displayed in Figure 5.8. The unicast method has the lowest average response time at approximately 59ms for networks with 10 clients and 116ms for networks with 200 clients. The traditional multicast protocol is about 1.3 times slower than unicast. SMP performs slightly slower than multicast with about 10% higher average response time. The response times for different variations of the tc-SMP methods are fairly close, on average about 2.0 to 2.5 times higher than the unicast method. This represents an average increase of 15% in response time over multicast.

SMP and the two versions of tc-SMP have significant processing overhead at the server required to measure the similarity between messages and to aggregate the similar ones. Small additional processing time at intermediate nodes is required because midway routers need to split incoming SMP messages into multiple outgoing messages for next-hop routers. Similar overhead also occurs in traditional multicast routing but is slightly smaller at transitional routers.

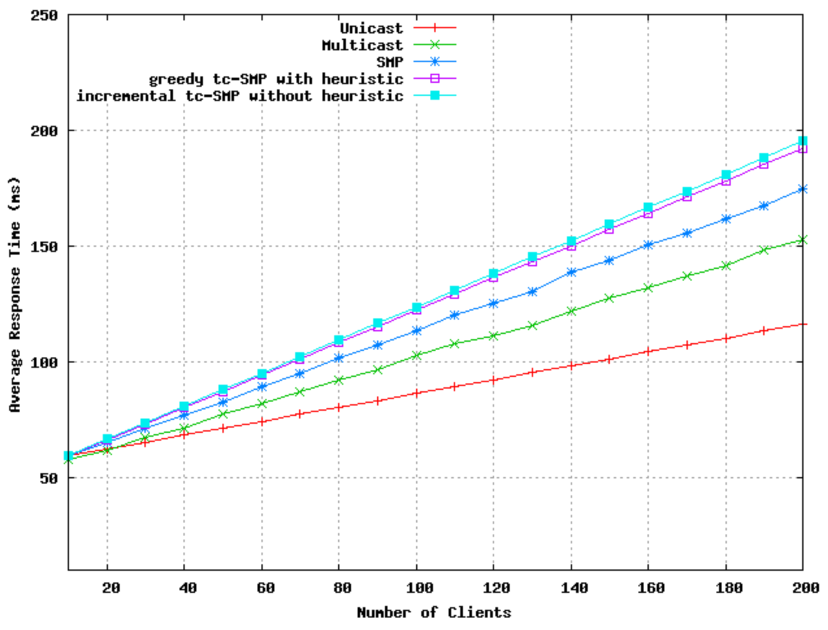


Figure 5.8: Average response time comparisons between different routing protocols

The performance penalty of tc-SMP over SMP is primarily its overhead in setting up the routing tree initially at the server. However, the difference in the average delays is not

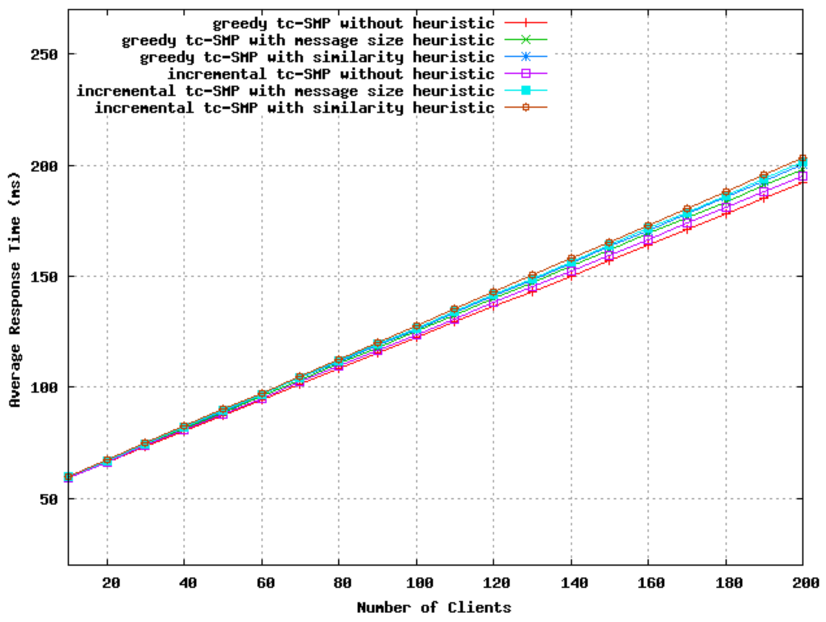


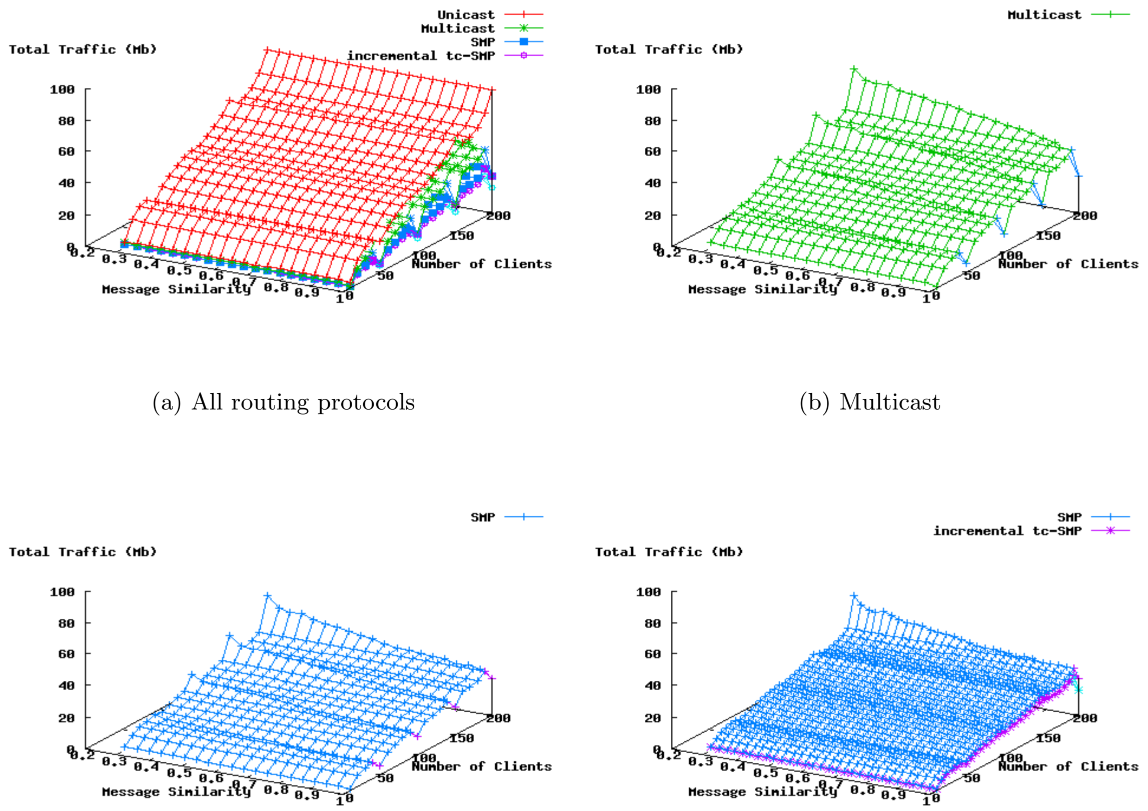
Figure 5.9: Average response time comparisons between different heuristics and non-heuristic tc-SMP algorithms

significant. For tc-SMP without heuristic, the average delay a client experiences ranges from 60ms to 195ms for networks of 10 clients to 200 clients. The corresponding results for SMP are 59.5ms to 175ms. On average, using the tc-SMP algorithms raises the average response by around 10% compared to SMP.

Figure 5.9 shows the response times for different variations of tc-SMP algorithms with and without heuristics. As shown in the graph, the *incremental* algorithm responds more slowly than the *greedy* one. This is because the *incremental* method needs to build an SMP tree first based on Dijkstra's algorithm, before incrementally building a tc-SMP tree. In addition, between the two heuristics used for selecting the order in which clients are added to a tc-SMP tree, the method that is based on the similarity between response messages takes a longer time. In the similarity-based heuristic method, the largest message is found first, then subsequent messages that have the greatest similarity with the first message will join a tc-SMP tree. Therefore, it is reasonable to expect that the similarity-based heuristic will have higher response time than the message size-based heuristic.

B) Maximum Optimality

This section presents an analysis of the network traffic reduction achieved by tc-SMP over SMP, multicast and unicast protocols in various message characteristic scenarios, to find the best variant of tc-SMP.



(a) All routing protocols (b) Multicast

(c) SMP (d) SMP vs. incremental tc-SMP

Figure 5.10: Total traffic comparisons between different protocols in different message similarity scenarios

The results presented in the previous section were obtained using a Zipf distribution of client requests. Here the behaviors of unicast, multicast, SMP and *incremental* tc-SMP for networks of different client request distributions are examined. The client request distribution is based on the overall similarity between all response messages ranging from 0.3 to 1 in this experiment. Additionally, for simplicity and ease of comparisons, the size of all response messages is constant at 40Kb. The experiment was carried out for networks of 10, 50, 100, 150 and 200 clients. The results for four protocols are displayed in Figure 5.10(a).

According to the 3D graph in Figure 5.10(a), incremental tc-SMP outperforms other protocols consistently over various network sizes and message similarity scenarios. When the similarity between all response messages is 1, the performance of SMP and multicast is similar, but incremental tc-SMP gives a performance gain of 25%. This reduction in traffic can be seen from the graph in Figure 5.10(d).

Figures 5.10(b) and 5.10(c) show the individual traffic results created by multicast and SMP respectively. It can be seen from these graphs that there is a trend of lower total traffic when the similarity between all messages is higher. The same trend also applies to incremental tc-SMP (Figure 5.10(d)). However, with the unicast method for the same network size (top graph in Figure 5.10(a)), the amount of traffic generated is the same regardless of the similarity between the messages.

The best reduction in total network traffic, 80%, of tc-SMP over unicast is achieved when all response messages are identical. In this scenario, multicast and SMP produce only 70% reduction in traffic over unicast. When the similarity between all the response messages is low (0.3), the performance gain of tc-SMP over unicast is 40% compared to 35% and 25% for SMP and multicast over unicast respectively.

5.6.3 Validation of the Results

In this section, the experimental results are validated by comparing them with the analytical results. Only incremental tc-SMP is examined here because the incremental approach outperforms the greedy approach and similar analysis can be applied to the greedy approach without difficulty. In Section 5.5.1, the best case scenario for *incremental* tc-SMP over SMP in total traffic was examined. In that scenario, SMP becomes unicast since there is no shared link between paths spanning all clients. Therefore, the upper boundary for incremental tc-SMP's theoretical total traffic result is the difference between unicast's theoretical total traffic (see Equation 4.15) and the maximum limit of *Diff_Traffic{SMP, tc-SMP(incremental)}* (see Expression 5.8). From this observation and Expression 5.3, the theoretical limits of the total traffic result for incremental tc-SMP is shown by the following:

$$\begin{aligned} \mathit{Total\_Traffic}\{unicast\} - (hN + 1 - h - N)size_{msg} &\leq \mathit{Total\_Traffic}\{\text{tc-SMP(incremental)}\} \\ &\leq \mathit{Total\_Traffic}\{SMP\} \end{aligned} \tag{5.15}$$

From Equation 5.14, the upper and lower limits of the theoretical average response time for incremental tc-SMP can be given as follows:

$$\begin{aligned} Average\_Response\{SMP\} &\leq Average\_Response\{\textit{tc-SMP (incremental)}\} \\ &\leq Average\_Response\{SMP\} + N \times n \times t_{path} \quad (5.16) \end{aligned}$$

The experimental results of incremental tc-SMP are compared to its boundary values based on Expressions 5.15 and 5.16, and presented in Figures 5.11 and 5.12. Table 5.2 outlines the values of the key parameters used for obtaining the graphs.

Parameter	**Value**
$size_{msg}$	40Kb
t_{path}	0.001ms
n	3 × N

Table 5.2: Assumptions of values used to obtain the theoretical boundaries in total traffic and average response time for the incremental tc-SMP algorithm

Variable t_{path} is the time it takes to determine if there exists a path from a node r to a client c and to compute the additional traffic introduced if c is added to a tree through r. One

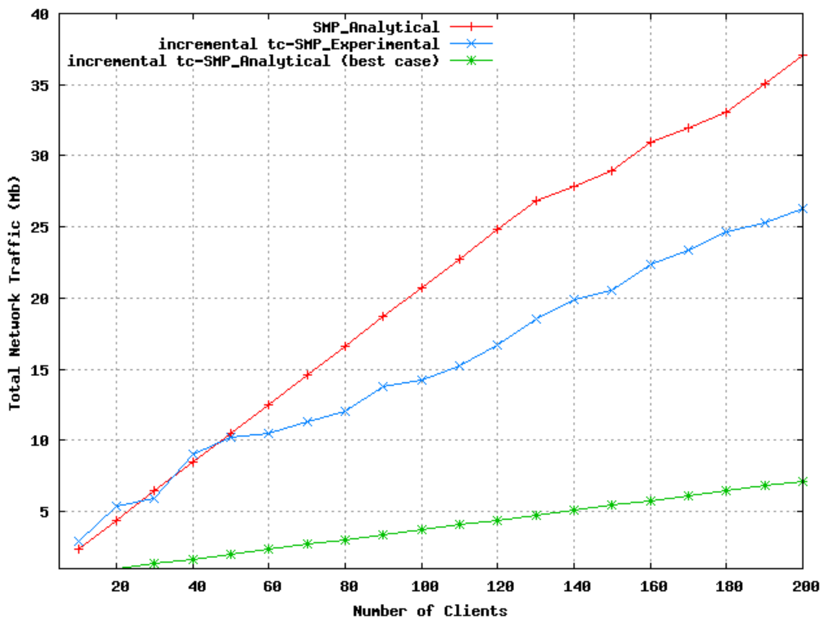


Figure 5.11: Analytical total network traffic analysis for incremental tc-SMP algorithm

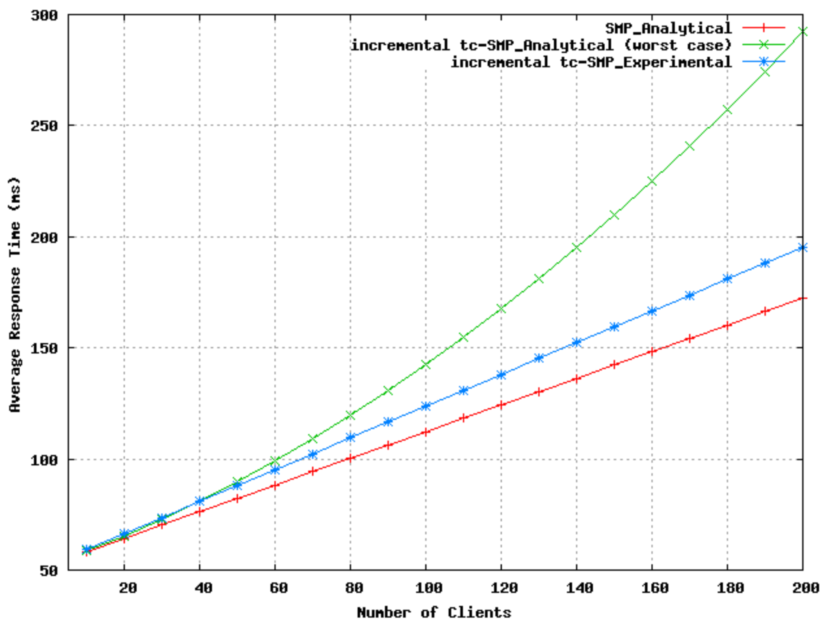


Figure 5.12: Analytical average response time analysis for incremental tc-SMP algorithm

important point to note is that not all nodes in a network have paths connecting to all clients. If such a path exists, the process of computing the additional cost is also straightforward because the similarity between a new message and messages to existing clients in a tree is already known in advance by the server from the initial aggregation process. Therefore, the size of the new aggregated message with the inclusion of the new client can be computed easily. For this reason, the experiments assumed that $t_{path} = 0.001ms$. The total number of nodes in a network is denoted by n, which is set to be 3 times of the client number, as an approximation.

From Figures 5.11 and 5.12, it is obvious that experimental total traffic and average response time results (middle lines in both graphs) of the *incremental* tc-SMP algorithm are within their analytical boundaries. Therefore, the evaluation of tc-SMP is validated by both analytical and experimental studies.

5.7 Discussion

This section presents a thorough analysis of the experimental results and discuss the advantages and disadvantages of tc-SMP over SMP.

Using tc-SMP, the traffic load is 4 or 5 times less than the traffic generated when individual original SOAP response messages are sent as unicasts. Tc-SMP reduces network bandwidth consumption of around 30% compared to SMP. A disadvantage of tc-SMP over SMP is that it requires additional time to build its own routing tree, which leads to an average response time increase of less than 10%.

As shown in Section 5.4.5, the tc-SMP routing algorithms perform in polynomial time, so the additional computation time is acceptable. The use of tc-SMP can be justified by traffic reduction whenever the increased response time is acceptable — from 3.5 up to 5 times reduction in traffic compared to under 2.5 times increase in average response time. Of course, results vary depending on network configurations.

This amount of delay is tolerable for many Web service applications, for example wireless communication among Intranet users, and personalized information retrieval over mobile networks. Tc-SMP represents a method for compressing size of messages in a network, thus it may be suitable for sensors network applications where bandwidth is limited and devices are constrained in power and battery life. Reducing the bandwidth consumption also benefits other applications by reducing traffic which is sent across those same links. The tc-SMP algorithm is particularly suitable in cases where the underlying networks are known to have multiple paths between nodes.

In addition, in high-speed wide-area networks, the link transmission delay is usually in microseconds, while the propagation delay may be close to milliseconds (much more greater) — so given a powerful processor, the trade-off in higher server processing overhead using tc-SMP over SMP is negligible. In wide-area networks propagation delay contributes the majority of the total delay.

5.8 Summary

This chapter examined the traffic-constrained SMP, called tc-SMP, problem where SMP messages are routed along paths that result in the least total network traffic. Two algorithms were proposed, both being extensions of the SMP algorithm presented in Chapter 4. Tc-SMP algorithms are an improvement over SMP for SOAP multicast traffic. They differ in the way that a source-routing algorithm (*greedy* or *incremental* approach) is used in tc-SMP for delivering aggregated messages along paths to introduce the minimal network traffic, instead of using the OSPF routing method which is used in SMP.

The *greedy* tc-SMP algorithm is the simpler approach of the two — every client is added to a tree along a path that introduces the least traffic. The *incremental* algorithm has similar objective, but it builds a tc-SMP tree gradually from a shortest path SMP tree — as such it can ensure that it outperforms or performs at least as well as SMP.

The problem of building a traffic-constrained similarity-based SOAP multicast tree is NP-complete. The proposed algorithms provide a good solution and operate in polynomial time with a complexity of $O(n(m + nlogn))$. On average, the incremental tc-SMP algorithm takes a little more time (less than 2% increase in average response time) time than the greedy algorithm, but provides a greater reduction (around 5%) in total traffic.

Simulations have shown that tc-SMP algorithms reduce further traffic generated over a network, by around 30%, when compared to SMP. The performance trade-off of tc-SMP over SMP is an increase in average response time of less than 10%. As the number of clients increases, the network traffic under tc-SMP is reduced even more than SMP, while the performance penalty is comparatively small. Two heuristic methods were also implemented for the tc-SMP algorithms. The heuristic based on the similarity between messages to new clients and messages to existing clients gives a gain of around 3% over tc-SMP without any heuristic. The heuristic based on message size achieves negligible performance gain.

Chapter 6

Conclusion

Improving SOAP performance is important to allow the widespread deployment of Web services in different environments, especially in low bandwidth networks. To cope with the high network traffic created by SOAP messages and low bandwidth availability, new techniques must be used to reduce the amount of traffic so that SOAP can perform well in limited bandwidth environments. This research thesis proposed a similarity-based SOAP multicast protocol to reduce network traffic. An alternative SOAP-over-UDP binding was also examined.

6.1 Key Contributions

This thesis presents three main contributions.

A) SOAP Binding Benchmark

Firstly, a benchmark suite of SOAP bindings in wireless environments was proposed and implemented. The experimental results reveal that HTTP binding, which is the most often used in Web service applications, exposes high protocol overhead in wireless environments due to the slow start, connection establishments and packet acknowledgements. On the other hand, UDP binding has lower overhead, hence reduces response time and increases total throughput. Experimental results show that UDP binding can reduce message size by between 30% to 50% compared to HTTP binding. Such reduction leads to a huge saving in transmission cost (50%–75%) and a significant increase in throughput (40%–78%).

B) Similarity-based SOAP Multicast Protocol

Another contribution is a novel SOAP multicast technique, called SMP, which exploits on the similarity of SOAP messages to efficiently multicast SOAP messages to a large number of clients requesting similar operations of the same service. SMP takes advantage of the similarity of SOAP messages' data structures and combines messages of similar content in one message. In this way, common data structures or values are not repeated and are sent only once, resulting in reduced network traffic. While previous studies utilised this feature of SOAP to improve the processing time of serialisation and deserialisation, those results were noticeable only at the server or client side and did not affect the network. Through experiments, it was found that the total traffic over a network can be reduced by 35% to 70% if SMP is used instead of unicast.

A model was also presented to measure the similarity of outgoing SOAP messages, so that the server can determine if messages have enough common XML structure or payload values to compensate the cost required to process SMP aggregated messages. In addition, the proposed SOAP indexing technique enables fast merging and splitting of SMP messages. Instead of a full SOAP message, an indexed version of the message, which is composed of data type IDs referenced back to the WSDL document, positions of each node in the SOAP message and the node values, is sent. SOAP messages' index representation further reduces the message size because of the omission of full XML tag names. The index representation also improves the grouping of common and distinctive data in SMP messages.

C) SOAP Multicast Protocol to Minimize SOAP Network Traffic

The third contribution of this thesis is new routing algorithms for traffic-constrained SMP (called tc-SMP), so that the network traffic can be minimised. Two algorithms: greedy and incremental approaches are proposed and extended from SMP to solve the tc-SMP problem. Tc-SMP algorithms offer better performance than SMP for SOAP multicast traffic. Tc-SMP differs from SMP in the way that a source-routing algorithm is used under tc-SMP for delivering aggregated messages instead of OSPF. Test results show that tc-SMP outperforms SMP by 10–25% further reduction in network traffic.

The greedy tc-SMP approach aims to minimize the total traffic in a tc-SMP tree every time a new client is added to the tree. However, this approach does not guarantee that the greedy tc-SMP always perform better than the original SMP. The incremental approach firstly establishes an SMP tree and iteratively improves this SMP tree. Using this approach, incremental tc-SMP never underperforms SMP. Experiments show that incremental tc-SMP is better than greedy tc-SMP in general but there is a small cost in response time. Two heuristics were also proposed for the tc-SMP algorithms: one is based on message size and the other is based on similarity with previous messages. The heuristic based on the similarity between messages to new clients and messages to existing clients gives a gain of around

3 percent over tc-SMP without any heuristic. The message size-based heuristic achieves negligible performance gain.

6.2 Research Achievements

Through an extensive set of experiments for the first contribution, it is shown that using HTTP or TCP binding for SOAP leads to significantly higher overhead than using the UDP binding. For a large number of requests of small to medium size messages, SOAP-over-UDP is about 4–5 times faster than SOAP-over-HTTP and around 3–3.5 times faster than SOAP-over-TCP. Specifically, the throughput of UDP binding is approximately 6 times higher for small messages, and 3 times higher for large messages, than the HTTP binding.

Detailed mathematical models of unicast, traditional IP multicast, SMP and tc-SMP were developed in Chapters 4 and 5. These models provide an insight into the properties of different routing protocols, and enable accurate comparisons to be made between them according to important measurement metrics such as total network traffic and average response time. Experimental evaluations have also been carried out to verify the accuracy of the theoretical models. The results obtained from the experiments fall into the boundaries of the analytical models, therefore the correctness of the models is confirmed.

SMP's main advantage is that it reduces the traffic generated over a network by 50–70% compared to unicast in large networks (over 100 clients) with large message payload (over 100Kb) at a cost of small increase in response time. There is a 3 time reduction in traffic compared to 1.5 time increase in response time — which is acceptable in Web service applications that handle a large amount of data in limited bandwidth environments, such as wireless networks, and do not have strong constraints on end-to-end delay.

The traffic-constrained similarity-based SOAP multicast problem is known to be NP-hard. The proposed greedy and incremental tc-SMP algorithms provide a good solution and operate in polynomial time with a complexity of $O(n(m + nlogn))$ where n is the number of nodes and m is the number of links in a network. Experiments have shown that tc-SMP algorithms outperform SMP, a reduction of around 30% in traffic compared to SMP, at a cost of 10% increase in response time. As the network size increases, the reduction in network traffic under tc-SMP increases even more than under SMP, while the performance penalty still remains comparatively small. On average, the incremental tc-SMP algorithm provides a greater reduction (around 5 percent) in total traffic, but takes a little more time (less than 2 percent increase in average response time) time than the greedy algorithm.

6.3 Future Work and Directions

The future work and research directions of SMP, tc-SMP and UDP binding techniques are discussed here.

An experiment on the use of UDP binding for SMP or tc-SMP protocol will be an interesting study. The low protocol overhead of UDP binding which is due to the absence of connection establish and tear-down processes and of strict flow control mechanism offers a promise to accelerate the overall processing time in SMP and tc-SMP. However, real experiments are needed to validate this theory.

Future work will involve researching better heuristic algorithms to further improve performance. Considerations of other quality of service parameters (such as delay bounds and bandwidth requirements) for each client may also be incorporated into the tc-SMP routing algorithm.

Another ambitious work plan is extending SMP and tc-SMP to support communication from different servers. Currently SMP or tc-SMP only supports aggregations of SOAP messages generated from the same service provider. Messages created from the same service have the same set of data type definitions in the WSDL service description, therefore merging and splitting of these messages are easier than doing the same process on messages that have different schemas. It is a challenge to find the syntactic similarity of messages from various providers. However if this can be done, and intermediate routers that are equipped with SMP layer can integrate messages from different sources together and sent only the union of those (all repeated sections are avoided), the network traffic will be considerably reduced. SOAP will become more dominantly viable to many business applications.

Lastly, when a service provider can provide different SOAP frameworks, conventional SOAP, SMP, tc-SMP, UDP binding and reliable UDP binding, a negotiation mechanism can be developed to allow clients to negotiate with the server their preferred SOAP representation depending on their performance requirements. For example, client applications that have a strict delay requirement and no specific constraint on bandwidth or reliability can request to use SOAP implementation that uses UDP binding. Applications that are constrained on low total network traffic and no requirement in response time could use SMP or tc-SMP in their communications. The negotiation uses a conventional SOAP message, so that two endpoints can revert to the SOAP message based Web Service communication, if they fail to negotiate.

Appendix A

WSDL Specification for the Stock Quote Service

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:tns="http://localhost/"
  targetNamespace="http://localhost/"
                  xmlns:wsdl="http://localhost/schema/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://localhost/">
      <s:element name="GetStockQuote" id="GSQ">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
     <s:element name="StockQuoteResponse" id="SQR">
        <s:complexType>
          <s:sequence>
            <s:element  minOccurs="0" maxOccurs="1" name="ArrayOfStockQuote"
    type="tns:ArrayOfStockQuote"/>
          </s:sequence>
        </s:complexType>
      </s:element>
```

```
<s:element name="GetQuoteAndStatistic" id="GQAS">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:element>
   <s:element name="QuoteAndStatisticResponse" id="QASR">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="ArrayOfQuoteAndStatistic"
         type="tns:ArrayOfQuoteAndStatistic"/>
        </s:sequence>
      </s:complexType>
    </s:element>
<s:element name="GetFullQuote" id="GFQ">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="FullQuoteResponse" id="GFQR">
      <s:complexType>
        <s:sequence>
          <s:element  minOccurs="0" maxOccurs="1" name="ArrayOfFullQuote"
          type="tns:ArrayOfFullQuote"/>
        </s:sequence>
      </s:complexType>
    </s:element>
   <s:element name="GetCompanyInfo" id="GCI">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="CompanyInfoResponse" id="CIR">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="ArrayOfCompanyInfo"
```

```
 type="tns:ArrayOfCompanyInfo"/>
        </s:sequence>
      </s:complexType>
    </s:element>
<s:element name="GetMarketInfo" id="GMI">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="MarketInfoResponse" id="MIR">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="ArrayOfMarketInfo"
     type="tns:ArrayOfMarketInfo"/>
        </s:sequence>
      </s:complexType>
    </s:element>
<s:simpleType name="Symbol" id ="Sbl">
     <s:restriction base="string">
       <s:length value="3"/>
     </s:restriction>
    </s:simpleType>
  <s:simpleType name="Price" id="P">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
<s:simpleType name="LastUpdated" id="LU">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
    <s:simpleType name="Change"  id ="Chg">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
<s:simpleType name="OpenPrice"  id ="OP">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
   <s:simpleType name="CompanyName" id ="CN">
```

```
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
   <s:simpleType name="ExDividendDate"  id ="EDD">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
<s:simpleType name="DividendYield"  id ="DY">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
<s:simpleType name="Volume"  id ="Vol">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
   <s:simpleType name="DayHighPrice"  id ="DHP">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
<s:simpleType name="DayLowPrice"  id ="DLP">
     <s:restriction base="string">
     </s:restriction>
    </s:simpleType>
    <s:complexType name="ArrayOfStockQuote" id="AOSQ">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="StockQuote"
       type="tns:StockQuote"/>
      </s:sequence>
    </s:complexType>
    <s:complexType name="ArrayOfQuoteAndStatistic" id="AOQS">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="QuoteAndStatistic"
           type="tns:QuoteAndStatistic"/>
      </s:sequence>
    </s:complexType>
    <s:complexType name="ArrayOfFullQuote" id="AOFQ">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="FullQuote"
       type="tns:FullQuote"/>
      </s:sequence>
    </s:complexType>
```

```
      <s:complexType name="ArrayOfCompanyInfo" id="AOCI">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="ComapnyInfo"
         type="tns:ComapnyInfo"/>
        </s:sequence>
      </s:complexType>
      <s:complexType name="ArrayOfMarketInfo" id="AOMI">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="MarketInfo"
type="tns:MarketInfo"/>
        </s:sequence>
      </s:complexType>
      <s:complexType name="StockQuote" id="SQ">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="tns:Symbol"/>
          <s:element minOccurs="0" maxOccurs="1" name="QuoteInfo" type="tns:QuoteInfo"/>
        </s:sequence>
      </s:complexType>
      <s:complexType name="QuoteAndStatistic" id="QAS">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="tns:Symbol"/>
          <s:element minOccurs="0" maxOccurs="1" name="QuoteInfo" type="tns:QuoteInfo"/>
          <s:element minOccurs="0" maxOccurs="1" name="Statistic" type="tns:Statistic"/>
        </s:sequence>
      </s:complexType>
      <s:complexType name="FullQuote" id="FQ">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="tns:Symbol"/>
          <s:element minOccurs="0" maxOccurs="1" name="QuoteInfo" type="tns:QuoteInfo"/>
          <s:element minOccurs="0" maxOccurs="1" name="Statistic" type="tns:Statistic"/>
          <s:element minOccurs="0" maxOccurs="1" name="CompanyInfo" type="tns:CompanyInfo"/>
        </s:sequence>
      </s:complexType>
      <s:complexType name="QuoteInfo" id="QI">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Price" type="tns:Price"/>
          <s:element minOccurs="0" maxOccurs="1" name="LastUpdated" type="tns:LastUpdated"/>
        </s:sequence>
      </s:complexType>
      <s:complexType name="Statistic" id="Stt">
        <s:sequence>
```

```
          <s:element minOccurs="0" maxOccurs="1" name="Change" type="tns:Change"/>
          <s:element minOccurs="0" maxOccurs="1" name="OpenPrice" type="tns:OpenPrice"/>
        </s:sequence>
      </s:complexType>
      <s:complexType name="CompanyInfo" id="CI">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="tns:Symbol"/>
          <s:element minOccurs="0" maxOccurs="1" name="CompanyName" type="tns:CompanyName"/>
          <s:element minOccurs="0" maxOccurs="1" name="ExDividendDate"
             type="tns:ExDividendDate"/>
          <s:element minOccurs="0" maxOccurs="1" name="DividendYield"
 type="tns:DividendYield"/>
        </s:sequence>
      </s:complexType>
      <s:complexType name="MarketInfo" id="MI">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Symbol" type="tns:Symbol"/>
          <s:element minOccurs="0" maxOccurs="1" name="Volume" type="tns:Volume"/>
          <s:element minOccurs="0" maxOccurs="1" name="DayHighPrice"
 type="tns:DayHighPrice"/>
          <s:element minOccurs="0" maxOccurs="1" name="DayLowPrice" type="tns:DayLowPrice"/>
        </s:sequence>
      </s:complexType>
     <s:element name="ArrayOfStockQuote"  nillable="true" type="tns:ArrayOfStockQuote"/>
 <s:element name="ArrayOfQuoteAndStatistic"  nillable="true"
type="tns:ArrayOfQuoteAndStatistic"/>
   <s:element name="ArrayOfFullQuote"  nillable="true" type="tns:ArrayOfFullQuote"/>
    <s:element name="ArrayOfCompanyInfo"  nillable="true" type="tns:ArrayOfCompanyInfo"/>
     <s:element name="ArrayOfMarketInfo" nillable="true" type="tns:ArrayOfMarketInfo"/>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetStockQuoteSoapIn">
    <wsdl:part name="parameters" element="tns:GetStockQuote"/>
  </wsdl:message>
  <wsdl:message name="GetStockQuoteSoapOut">
    <wsdl:part name="parameters" element="tns:StockQuotesResponse"/>
  </wsdl:message>
  <wsdl:message name="GetStockQuoteHttpGetIn">
    <wsdl:part name="Symbol" type="s:string"/>
  </wsdl:message>
  <wsdl:message name="GetStockQuoteHttpGetOut">
```

```
    <wsdl:part name="Body" element="tns:ArrayOfStockQuote"/>
  </wsdl:message>
  <wsdl:message name="GetStockQuotesHttpPostIn">
    <wsdl:part name="Symbol" type="s:string"/>
  </wsdl:message>
  <wsdl:message name="GetStockQuotesHttpPostOut">
    <wsdl:part name="Body" element="tns:ArrayOfStockQuote"/>
  </wsdl:message>
  <wsdl:portType name="StockQuoteSoap">
    <wsdl:operation name="GetStockQuote">
      <wsdl:input name="GetStockQuote" message="tns:GetStockQuoteSoapIn"/>
      <wsdl:output name="GetStockQuote" message="tns:GetStockQuoteSoapOut"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="StockQuoteSoap" type="tns:StockQuoteSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="GetStockQuote">
      <soap:operation soapAction="http://localhost/StockQuote" style="document"/>
      <wsdl:input name="GetStockQuote">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="GetQuotes">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="StockQuote">
    <wsdl:port name="StockQuoteSoap" binding="tns:StockQuoteSoap">
      <soap:address location="http://localhost/StockQuote"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Appendix B

SMP Message Schema

```
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="SMP:Header" type="SMPHeaderType"/>
 <xs:element name="SMP:Body" type="SMPBodyType"/>
 <xs:complexType name="SMPHeaderType">
  <xs:sequence>
   <!-- indexing the address of a client -->
   <xs:element name="c" type="cType"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="cType">
  <!-- the index id of the client-->
  <xs:attribute name="id" type="xs:unsignedInt" use="required"/>
   <!-- the actual address of the client in the network -->
  <xs:attribute name="uri" type="xs:string" use="required"/>
 </xs:complexType>
 <xs:complexType name="SMPBodyType">
  <xs:sequence>
   <xs:element name="Common" type="CommonType"/>
   <xs:element name="Distinctive" type="DistType"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="CommonType">
  <xs:sequence>
   <xs:element name="part" type="partType" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="partType">
```

```
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <!-- refer to another part defined elsewhere -->
  <xs:attribute name="refer" type="xs:string" use="optional"/>
  <xs:sequence>
  <!-- This element stores the tag value -->
  <xs:element name="v" type="vType" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="vType">
  <!-- refer to an element id defined in the WSDL -->
  <xs:attribute name="eRef" type="xs:string" use="required"/>
  <!-- actual value of a tag, which can be empty-->
  <xs:attribute name="val" type="xs:string" use="optional"/>
</xs:complexType>
<xs:complexType name="DistType">
 <xs:sequence>
  <xs:element name="DPart" type="DPartType" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="DPartType">
 <xs:sequence>
  <!-- the node position for specified clients-->
  <xs:element name="cPos" type="cPosType"  maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="cPosType">
 <!-- the list of client indices-->
 <xs:attribute name="Ids" type="xs:string" use="required"/>
  <!-- the position of a node in messages addressed to the clients -->
 <xs:attribute name="pos" type="xs:string" use="required"/>
</xs:complexType></xs:schema>
```

Appendix C

A Sample SMP Message

An SMP Message, called SMP_2, aggregating $Soap_1$, $Soap_3$ and $Soap_4$ messages as explained in Section 4.6.2 of Chapter 4, is shown below.

```
<soap:Envelope>
 <soap:Header>
  Next-hop router's address
 </soap:Header>
 <soap:Body>
  <SMP:Header>
   <c id='1', uri='URI1'/>
   <c id='2', uri='URI2'/>
   <c id='3', uri='URI3'/>
  </SMP:Header>
  <SMP:Body>
   <Distinctive>
    <DPart>
      <cPos cIds ='1,2' pos='0'/>
      <part refer='p1'/>
    </DPart>
    <DPart>
      <cPos cIds ='1' pos='1'/>
      <part>
       <v eRef= 'AOSQ' val='2'/>
      <part/>
    </DPart>
    <DPart>
      <cPos cIds ='2' pos='1'/>
      <part>
```

```
       <v eRef= ‘AOSQ’ val=‘3’/>
      <part/>
    </DPart>
    <DPart>
      <cPos cIds =‘1,2’ pos=‘2.1’/>
      <part refer=‘p2’/>
    </DPart>
    <DPart>
      <cPos cIds =‘1’ pos=‘2.2’>
      <cPos cIds =‘2’ pos=‘2.3’>
      <part refer=‘p3’>
    </DPart>
    <DPart>
     <cPos cIds =‘1’ pos=‘2.2’>
     <cPos cIds =‘2’ pos=‘2.3’>
     <part Id=‘p3’>
       <v eRef=‘SQ’ val=‘’/>
       <v eRef=‘Sbl’ val=‘BHP’/>
       <v eRef=‘QI’ val=‘’/>
       <v eRef=‘P’ val=‘24.52’/>
       <v eRef=‘LU’ val=‘24/01/2007 10:45am’/>
     </part>
    </DPart>
    <DPart>
      <cPos cIds =‘2’ pos=‘2.2’>
      <part Id=‘p4’>
       <v eRef=‘SQ’ val=‘’/>
       <v eRef=‘Sbl’ val=‘WIL’/>
       <v eRef=‘QI’ val=‘’/>
       <v eRef=‘P’ val=‘1.29’/>
       <v eRef=‘LU’ val=‘24/01/2007 10:45am’/>
      </part>
    </DPart>
   </Distinctive>
 <Common>
     <part Id=‘p1’>
       <v  eRef= ‘SQR’ val=‘’/>
     </part>
     <part Id=‘p2’>
       <v  eRef= ‘SQ’ val=‘’/>
       <v  eRef= ‘Sbl’  val=‘NAB’/>
       <v eRef='QI' val=''/>
       <v eRef='P' val='28.56'/>
       <v eRef='LU' val='24/01/2007 10:15am'/>
     </part>
   </Common>
  </SMP:Body>
 </soap:Body>
</soap:Envelope>
```

Bibliography

N. Abu-Ghazaleh and M. Lewis. Differential deserialization for optimized soap performance. In *Proceedings of the 2005 ACM/IEEE Conference on Super-Computing*, pages 21–31, Seattle, WA, USA, November 2005.

N. Abu-Ghazaleh, M. Lewis, and M. Govindaraju. Differential serialization for optimized SOAP performance. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, pages 55–64, Honolulu, Hawaii, USA, 2004. IEEE Computer Society.

N. Abu-Ghazaleh, M. Govindaraju, and M. Lewis. Optimizing Performance of Web Services with Chunk-Overlaying and Pipelined-Send. In *Proceedings of the International Conference on Internet Computing*, pages 482–485, Las Vegas, NV, USA, June 2004.

ADT. A little action on Java-ready phones, 2005.
URL: http://www.adtmag.com/article.asp?id=11017.

M. Allman, V. Paxson, and W. Stevens. *TCP Congestion Control.* RFC 2581 Internet EngineeringTask Force (IETF), April 1999.
URL: http://tools.ietf.org/html/rfc2581.

L. M. Andreas Laux. XUpdate working draft, 2000. URL http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html.

R. Angelen. gSOAP: C/C++ Web Services and Clients, 2003.
URL: http://www.cs.fsu.edu/~engelen/soap.html.

Apache Software Foundation. *Web Services Axis*, 2007a.
URL: http://ws.apache.org/axis.

Apache Software Foundation. *Apache Tomcat*, 2007b.
URL: http://tomcat.apache.org/.

N. Apte, K. Deutsch, and R. Jain. Wireless SOAP: optimizations for mobile wireless Web services. In *Special interest tracks and posters of the 14th International Conference on World Wide Web*, pages 1178–1179, New York, NY, USA, 2005. ACM Press.

D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and principles of internet traffic engineering. RFC 3272, Internet Engineering Task Force, 2002. URL http://www.ietf.org/rfc/rfc3272.txt.

V. Bansal and A. Dalton. A performance analysis of Web services on wireless PDA. Research note, Duke University, Department of Computer Science, Durham, NC, USA, 2002.
URL: http://www.cs.duke.edu/vkb/advnw/project/PDAWebServices.pdf.

P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web*, 2(2):15–28, 1999.

BEA Systems Inc., Lexmark, Microsoft Corporation Inc., and Ricoh. *SOAP-over-UDP Specification*, 2004.
URL: http://ftpna2.bea.com/pub/downloads/SOAP-over-UDP.pdf.

R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

R. Bilorusets, D. Box, L. F. Cabrera, D. Davis, and D. Ferguson. *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)*, 2005.
URL: http://msdn.microsoft.com/library/en-us/dnglobspec/html/
WS-ReliableMessaging.pdf.

R. Boivie, N. Feldman, and C. Metz. Small group multicast: A new solution for multicasting on the Internet. *IEEE Internet Computing*, 4(3):75–79, 2000.

A. Boudani and B. Cousin. SEM: A new small group multicast routing protocol. In *Proceedings of the 10th International Conference on Telecommunications*, volume 1, pages 450–455, Tahiti, Papeete-French Polynesia, March 2003. IEEE Press.

A. Boudani, A. Guitton, and B. Cousin. Gxcast: Generalized explicit multicast routing protocol. In *Proceedings of the 9th International Symposium on Computers and Communications*, pages 1012–1017, Alexandria, Egypt, 2004. IEEE Press.

L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of the IEEE Conference on Computer Communications*, volume 1, pages 126–134, New York, NY, USA, March 1999.

D. Chakraborty, G. Chakraborty, C. Pornavalai, and N. Shiratori. Optimal routing for dynamic multipoint connection. *European Transactions on Telecommunication*, 10(2):183–189, March-April 1999.

D. Chakraborty, G. Chakraborty, and N. Shiratori. A dynamic multicast routing satisfying multiple QoS constraints. *International Journal of Network Management*, 13(5):321–335, 2003.

L. Chen and R. Nath. A framework for mobile business applications. *International Journal of Mobile Communications*, 2(4):368–381, 2004.

M. Chen, D. Zhang, and L. Zhou. Providing Web services to mobile users: the architecture design of an m-service portal. *International Journal of Mobile Communications*, 3(1):1–18, 2005.

S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Networks Magazine, Special Issue on Transmission and Distribution of Digital Video*, 12(6):64–79, 1998.

S. Chen and K. Nahrstedt. A distributed quality-of-service routing in ad-hoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1488–1505, August 1999.

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 246–255, Washington, DC, USA, 2002. IEEE Computer Society.

E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1.* World Wide Web Consortium (W3C), March 2001.
URL: http://www.w3.org/TR/wsdl.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, volume 2. MIT Press and McGraw-Hill, 2001.

E. Damiani, N. Lavarini, S. Marrara, B. Oliboni, D. Pasini, L. Tanca, and G. Viviani. The approxml tool demonstration. In *Proceedings of the 8th International Conference on Extending Database Technology*, pages 753–755, London, UK, 2002. Springer-Verlag.

A. Davis and D. Zhang. A comparative study of DCOM and SOAP. In *Proceedings of the 4th IEEE International Symposium on Multimedia Software Engineering*, pages 48–56, Newport Beach, CA, USA, December 2002. IEEE Computer Society.

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 407–413, Berlin, Germany, May 2002. IEEE Computer Society.

K. Devaram and D. Andresen. SOAP optimization via parameterized client-side caching. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 439–312, Berlin, Germany, May 2002. IEEE Computer Society.

M. Dewey. Dewey decimal classification. In *A Classification and Subject Index for Cataloguing and Arranging the Books and Pamphlets of a Library*. Kingsport Press, 2004. URL www.gutenberg.org/etext/12513.

M. Doar and I. Leslie. How bad is naive multicast routing? In *Proceedings of the 12th Annual Joint Conference of the IEEE Computer and Communications Societies, Networking: Foundation for the Future*, volume 1, pages 82–89, 1993.

C. Dorneles and et. al. Measuring similarity between collection of values. In *Proceedings of the 6th annual international workshop on Web information and data management*, pages 56–63, New York, USA, 2004. ACM Press.

R. Englander. Chapter 4: RPC-Style Services. In *Java and SOAP*. O'Reilly and Associates, 2002.

D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

S. Fell. *PocketSOAP*, 2004.
URL: http://www.pocketsoap.com/.

G. Feng, K. Makki, and N. Pissinou. Efficient implementations of bounded shortest multicast algorithm. In *Proceedings of the 11th International Conference on Computer Communications and Networks*, pages 312–317, 2002.

W. Fenner. RFC 2236 internet group management protocol, version 2, 1997. URL ftp://ftp.isi.edu/in-notes/rfc2236.txt.

M. Fernandez, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. *XQuery 1.0 and XPath 2.0 Data Model (XDM).* World Wide Web Consortium (W3C), January 2007.
URL: http://www.w3.org/TR/xpath-datamodel/.

C. Ferris and S. Williams. *SOAP Underlying Protocol Binding.* World Wide Web Consortium (W3C), October 2001. URL http://www.w3.org/2000/xp/Group/1/10/12/Binding_Framework_Proposal.

G. Fox and S. Pallickara. JMS compliance in the Narada event brokering system. In *Proceedings of the 2002 International Conference on Internet Computing*, volume 2, pages 391–397, 2002.

P. Ganesan, H. Garcia-Molina, and J. Widom. Exploiting hierarchical domain structure to compute similarity. *ACM Transactions on Information Systems*, 21(1):64–93, 2003a.

P. Ganesan, H. Garcia-Molina, and J. Widom. Exploiting hierarchical domain structure to compute similarity. *ACM Transactions on Information Systems*, 21(1):64–93, 2003b. ISSN 1046-8188.

D. Gisolfi. *Web services architect, Part 3: Is Web services the reincarnation of CORBA?* IBM, 2001.
URL: http://www.ibm.com/developerworks/webservices/library/ws-arc3/.

M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. van Engelen, and M. J. Lewis. Toward characterizing the performance of SOAP toolkits. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 365–372, Pittsburgh, USA, 2004. IEEE Computer Society.

E. A. Gryazin and O. Seppala. SOAP and CORBA productivity comparison for resource-limited mobile devices. In *Proceedings of the IASTED International Conference Software Engineering*, pages 707–712, Innsbruck, Austria, February 2004.

M. Gudgin, N. Mendelsohn, M. Nottingham, and H. Ruellan. *SOAP Message Transmission Optimization Mechanism.* World Wide Web Consortium (W3C), January 2005a.
URL: http://www.w3.org/TR/soap12-mtom/.

M. Gudgin, N. Mendelsohn, M. Nottingham, and H. Ruellan. *XML-binary Optimized Packaging.* World Wide Web Consortium (W3C), January 2005b.
URL: http://www.w3.org/TR/xop10/.

M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen. *SOAP Version 1.2 Part 1: Messaging Framework.* World Wide Web Consortium (W3C), 2nd edition, April 2007.
URL: http://www.w3.org/TR/soap12-part1/.

G. Guido and B. Ralf. Performance of mobile Web service access using the wireless application protocol (WAP). In *Proceedings of the 5th World Wireless Congress*, pages 427–432, San Francisco, CA, USA, May 2004.

L. Guo and I. Matta. QDMR: An efficient QoS dependent multicast routing algorithm. In *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium*, pages 213–222, Vancouver, Canada, June 1999. IEEE Computer Society.

B. Han, W. Jia, J. Shen, and M.-C. Yuen. Context-awareness in mobile web services. In *Parallel and Distributed Processing and Applications*, pages 519–528. Springer, 2004.

S. Haustein. *kXML*, 2005.
URL: http://kxml.sourceforge.net/.

S. Haustein and J. Seigel. kSOAP: An open source SOAP for the kVM, 2003.
URL: http://ksoap2.sourceforge.net/.

M. Head, M. Govindaraju, A. Slominski, P. Liu, N. Abu-Ghazaleh, R. Engelen, K. Chiu, and M. Lewis. A benchmark suite for SOAP-based communication in grid Web services. In *Proceedings of International Conference for High Performance Computing, Networking, and Storage*, pages 19–31, Seattle WA, USA, November 2005.

M. Horstmann and M. Kirtland. *DCOM Architecture.* Microsoft Corporation Inc, 1997.
URL: http://msdn2.microsoft.com/en-us/library/ms809311.aspx.

F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem.* Elsevier, North-Holland, 1992.

P. Jaccard. Jaccard index. *Bulletin del la Société Vaudoisedes Sciences Naturelles*, 37:241–272, 1901.

C. Kohlhoff and R. Steele. Evaluating SOAP for high performance business applications: Real-time trading systems. In *Proceedings of the 12th International conference on World Wide Web*, pages 872–880, Budapest, Hungary, 2003.

V. Kompella, J. Pasquale, and G. Polyzos. Two distributed algorithms for multicasting multimedia information. In *Proceedings of the 2nd International Conference on Computer Communications and Networks*, pages 343–349, San Diego, California, USA, 1993.

K. Y. Lai, T. K. A. Phan, and Z. Tari. Efficient soap binding for mobile web services. In *Proceedings of the 30th IEEE Conference on Local Computer Networks*, pages 218–225, Sydney, Australia, 2005. IEEE Computer Society.

S. Lee and G. Fox. Wireless reliable messaging protocol for web services (WS-WRM). In *Proceedings of the IEEE International Conference on Web Services*, pages 350–357, San Diego, CA, USA, 2004. IEEE Computer Society.

V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.

A. Levitin. *Introduction to the design and analysis of algorithms.* Addison Wesley, 2nd edition, 2007.

W. Liang and H. Yokota. LAX: An efficient approximate XML join based on clustered leaf nodes for XML data integration. In *Database: Enterprise, Skills and Innovation*, volume 3567, pages 82–97. Springer, 2005.

W. Liang and H. Yokota. A path-sequence based discrimination for subtree matching in approximate xml joins. In *Proceedings of the 22nd International Conference on Data Engineering Workshops*, page 116, Washington, DC, USA, 2006. IEEE Computer Society.

X. Liu and R. Deters. An efficient dual caching strategy for Web service-enabled PDAs. In *Proceedings of the 22nd Annual ACM symposium on Applied computing*, pages 788–794, Seoul, Korea, 2007. ACM Press.

J. Lu, T. W. Ling, C.-Y. Chan, and T. Chen. From region encoding to extended Dewey: on efficient processing of XML twig pattern matching. In *Proceedings of the 31st international conference on Very large data bases*, pages 193–204. VLDB Endowment, 2005.

W. Lu, K. Chiu, and D. Gannon. Building a generic SOAP framework over binary XML. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, pages 195–203, Paris, France, June 2006.

Y. Ma and R. Chbeir. Content and structure based approach for XML similarity. In *Proceedings of the 5th International Conference on Computer and Information Technology*, pages 136–140, Shanghai, China, September 2005. IEEE Computer Society.

A. Maedche and S. Staab. Comparing ontologies-similarity measures and a comparison study. Internal report no. 408, Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany, 2001.

S. Makino, M. Tatsubori, K. Tamura, and Y. Nakamura. Improving WS-security performance with a template-based approach. In *Proceedings of the IEEE International Conference on Web Services*, pages 581–588, Orlando, Florida, USA, 2005.

D. S. Marshak. *Sun Java System Platform and Architecture and Java Web Services Infrastructure Enables Easy Access to Government Services.* Sun Microsystems Inc., February 2004.
URL: http://www.sun.com/service/about/success/gta_p2.html.

J. McCarthy. *Reap the benefits of document style Web services.* IBM, June 2002.
URL: http://www.ibm.com/developerworks/webservices/library/ws-docstyle.html.

Microsoft Corporation. *.NET Compact Framework*, 2007.
URL: http://msdn2.microsoft.com/en-us/netframework/aa497273.aspx.

Microsoft Corporation. *Web Service Enhancement*, 2006.
URL: http://msdn2.microsoft.com/en-us/webservices/aa740663.aspx.

Microsoft Corporation Inc. *Thomson Financial: Cutting-Edge Financial Software Puts Asset Managers in Control*, November 2006.
URL: http://www.microsoft.com/casestudies/casestudy.aspx?casestudyid=200352.

Microsoft Corporation Inc. *Danske Bank: Danish Bank Uses Visual Studio .NET, Web Services to Generate New Revenue Sources*, February 2003.
URL: http://www.microsoft.com/casestudies/search.aspx?keywords=danske.

R. Mitchell. *Web Services on Mobile Devices*, June 2006.
URL: http://itmanagement.earthweb.com/entdev/article.php/3612721.

N. Mitra and et al. *SOAP Version 1.2 Part 0: Primer.* World Wide Web Consortium (W3C), 2nd edition, April 2007.
URL: http://www.w3.org/TR/soap12-part0/.

M. Mokbel, W. Elhaweet, and M. Elderini. An efficient algorithm for shortest path multicast routing under delay and delay variation constraints. In *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 190–196, Vancouver, Canada, July 2000. Enformatika.

A. Mouat. XML diff and patch utilities, 2002. URL http://diffxml.sourceforge.net/docs/docs.html.

A. Ng, S. Chen, and P. Greenfield. An evaluation of contemporary commercial SOAP implementations. In *Proceedings of the 5th Australasian Workshop on Software and System Architectures (AWSA)*, pages 64–71, Melbourne, Australia, 2003.

A. Ng, P. Greenfield, and S. Chen. A study of the impact of compression and binary encoding on SOAP performance. In *Proceedings of the 6th Australasian Workshop on Software and System Architectures*, pages 46–56, Brisbane, Australia, March 2005.

OASIS. *UDDI.* Organization for the Advancement of Structured Information Standards, 2006a. URL http://www.uddi.org/.

OASIS. *Web Services Reliable Messaging TC: WS-Reliability 1.1.* Organization for the Advancement of Structured Information Standards, 2004.
URL: http://docs.oasis-open.org/wsrm/2004/06/WS-Reliability-CD1.086.pdf.

OASIS. *OASIS Web Services Security (WSS) TC.* Organization for the Advancement of Structured Information Standards, November 2006b.
URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

S. Oh and G. Fox. HHFR: A new architecture for mobile Web services: Principles and implementations. Technical report, Indiana University, Community Grids Laboratory, Bloomington, IN, USA, 2005. URL http://grids.ucs.indiana.edu/ptliupages/publications/HHFR_ohsangy.pdf.

S. Oh, D. Lee, and S. Kumara. Web service planner (WSPR): An effective and scalable Web service composition algorithm. *International Journal of Web Services Research*, 4(1):1–23, 2007.

C. Oliveira, P. Pardalos, and M. Resende. Optimization problems in multicast tree construction. In *Handbook of Optimization in Telecommunications*, pages 701–733. Kluwer, Dordrecht, 2005.

OMG. *CORBA Basics*. The Object Management Group, 2001.
URL: http://www.omg.org/gettingstarted/corbafaq.htm.

OMG. *IIOP: OMG's Internet Inter-ORB Protocol - A Brief Description*. The Object Management Group, 2007.
URL: http://www.omg.org/news/whitepapers/iiop.htm.

P. Paul and S. Raghavan. Survey of QoS routing. In *Proceedings of the 15th International Conference on Computer Communication*, pages 50–75, Mumbai,India, August 2002. International Council for Computer Communication.

D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, pages 49–60, Seattle, Washington, USA, March 2001.

K. A. Phan, Z. Tari, and P. Bertok. A benchmark on SOAP's transport protocols performance for mobile applications. In *Proceedings of the 21st Annual ACM Symposium on Applied computing*, pages 1139–1144, Dijon, France, April 2006a. ACM Press.

K. A. Phan, Z. Tari, and P. Bertok. Optimizing Web services performance by using similarity-based multicast protocol. In *Proceedings of the 4th European Conference on Web Services*, pages 119–128, Zurich, Switzerland, December 2006b.

K. A. Phan, P. Bertok, A. Fry, and C. Ryan. Minimal traffic-constrained similarity-based SOAP multicast. In *Proceedings of the 9th International Symposium on Distributed Objects, Middleware, and Applications (to appear)*, Algarve, Portugal, November 2007a.

K. A. Phan, Z. Tari, and P. Bertok. eSMP: A multicast protocol to minimize SOAP network traffic in low bandwidth environments. In *Proceedings of the 32th Annual IEEE Conference on Local Computer Networks (to appear)*, Dublin, Ireland, October 2007b.

R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

V. Radha, V. Gulati, and A. Pujari. Efficient multicast e-services over APPCAST. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, volume 2, pages 331–338, Las Vegas, NV, USA, April 2004. IEEE Computer Society.

K. Ross and J. Kurose. *Connectionless Transport: UDP*, 2000.
URL: http://www-net.cs.umass.edu/kurose/transport/UDP.html.

A. Roy. OSPF Version 2. RFC 1247, Internet Engineering Task Force (IETF), 1991. URL http://www.ietf.org/rfc/rfc1247.txt.

H. Salama and D. Reeves. Evaluation of multicast routing algorithms for real-time communication on high-speed networks. *IEEE Journal on Selected Areas of Communication*, 15 (3), April 1997.

R. Shah, Z. Ramzan, and R. Dendukuri. Efficient dissemination of personalized information using content-based multicast. *IEEE Transactions on Mobile Computing*, 3(4):394–408, 2004. Senior Member-Ravi Jain and Member-Farooq Anjum.

A. Shaikh and K. Shin. Destination-driven routing for low-cost multicast. *IEEE Journal of Selected Areas in Communications*, 15(3):373–381, 1997.

M.-K. Shin, Y.-J. Kim, K.-S. Park, and S.-H. Kim. Explicit Multicast Extension (Xcast+) for efficient multicast packet delivery. *ETRI Journal*, 23(4):202–204, December 2001.

Sun Microsystems Inc. *Java 2 Platform, Micro Edition - J2ME*, 2004a.
URL: http://java.sun.com/javame/index.jsp.

Sun Microsystems Inc. *J2ME Web Services APIs (WSA), JSR 172*, 2004b.
URL: http://java.sun.com/products/wsa/.

Sun Microsystems Inc. Mobile information device profile (MIDP); JSR 37, JSR 118, 2007.
URL: http://java.sun.com/products/midp.

Sun Microsystems Inc. Performance code samples and applications, 2004c.
URL: http://java.sun.com/performance/reference/codesamples/.

T. Suzumura, T. Takase, and M. Tatsubori. Optimizing Web services performance by differential deserialization. In *Proceedings of the IEEE International Conference on Web Services*, pages 185–192, Orlando, Florida, USA, 2005.

T. Takase and M. Tatsubori. Efficient Web services response caching by selecting optimal data representation. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 188–197, Toronto, Canada, 2004. IEEE Computer Society.

T. Takase, H. Miyashita, T. Suzumura, and M. Tatsubori. An adaptive, fast, and safe XML parser based on byte sequences memorization. In *Proceedings of the 14th International Conference on World Wide Web*, pages 692–701, Chiba, Japan, 2005. ACM Press.

D. B. Terry and V. Ramasubramanian. Caching XML Web Services for Mobility. *ACM Queue*, 1(3):70–78, May 2003.

M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller. Performance considerations for mobile Web services. *Computer Communications*, 27(11):1097–1105, July 2004.

A. Varga. *OMNet++ Discrete Event Simulation System*, 2006.
URL: http://www.omnetpp.org.

W3C. *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*. World Wide Web Consortium, 2nd edition, April 2007a.
URL: http://www.w3.org/TR/soap12-part2/.

W3C. *WAP Binary XML Content Format*. World Wide Web Consortium, 1999. URL http://www.w3.org/TR/wbxml/.

W3C. *Web Services Addressing (WS-Addressing)*. World Wide Web Consortium, August 2004.
URL: http://www.w3.org/Submission/ws-addressing/.

W3C. *Extensible Markup Language (XML)*. World Wide Web Consortium, May 2007b.
URL: http://www.w3.org/XML/.

B. Waxman. Routing of multiple connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1986.

C. Werner, C. Buschmann, and S. Fischer. Compressing soap messages by using differential encoding. In *Proceedings of the IEEE International Conference on Web Services*, pages 540–547, San Diego, CA, USA, 2004. IEEE Computer Society.

C. Werner, C. Buschmann, and F. Fischer. WSDL-driven SOAP compression. *International Journal of Web Services Research*, 2(1):18–35, 2005.

S. Whittle. *Case study: Amazon Web Services*, July 2007.
URL: http://www.computing.co.uk/computing/analysis/2193374/
case-study-amazon-web-services.

X. Yuan. Heuristics algorithms for multiconstrained quality-of-service routing. *IEEE/ACM Transactions on Networking*, 10(2):244–256, April 2002.

B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1366–1375, June 2002.

N. Zhang, S. K. Agrawal, and T. Ozsu. BlossomTree: Evaluating XPaths in FLWOR expressions. In *Proceedings of the 21st International Conference on Data Engineering*, pages 388–389, Los Alamitos, CA, USA, April 2005. IEEE Computer Society.

Q. Zhu, M. Parsa, and J. Garcia-Luna-Aceves. A source-based algorithm for delay-constrained minimum cost multicasting. In *Proceedings of the IEEE Conference on Computer Communications*, pages 377–385, 1995.

G. K. Zipf. *Human Behaviour and the Principle of Least-Effort.* Addison-Wesley, Cambridge MA, 1949.