SPADE
A Test Framework for SOAP Analysis in Dynamic Environments

Morten Avlesen[1], Skage Spjelkavik[1], Bjørn Vik[1], Frank T. Johnsen[2] and Trude H. Bloebaum[2]
[1]*Høgskolen i Oslo og Akershus (HiOA), Oslo, Norway*
[2]*Norwegian Defence Research Establishment (FFI), Kjeller, Norway*

Keywords: Web Services.

Abstract: We have developed a SOAP Analysis in Dynamic Environments (SPADE) test framework for conducting experiments with Web services communication optimizations. Our framework consists of a Web service and a client, a proxy translator module, a network emulator module, and an analysis module. The Web service and client pair is intended for evaluation, and its only task is to provide traffic for testing. The translator module translates the communication between the Web service and the client to other protocols. The network emulator module simulates different network conditions. Finally, the analysis module is used to measure the performance of the different protocols. The contribution of this paper is our easily extendable test framework for evaluating Web services transport protocols (and compression algorithms).

1 INTRODUCTION

Web services technology comprises a set of programming standards providing the means for machine-to-machine communication (W3C Working Group Note, 2004). Although Web services can be seen as just another middleware, vital differences exist. For instance, where other types of middleware are typically used within a local domain, Web services technology is also used as a middleware to connect such local domains over the Internet. Web services are also breaking ground in relation to standardization of middleware platforms with respect to language (XML), interface (WSDL), and message exchange (SOAP) (Taylor, 2005).

The most common transport binding for Web services is to use HTTP over TCP, which is one of several standardized transport bindings for SOAP messages (UDP and SMTP bindings also exist). One of the main reasons why the HTTP binding is the most common is the fact that the majority of tools available for Web service development support only this binding. This in turn means that in order to remain compatible with other service implementations, it is usually necessary to retain this transport binding as the communication interface between services and clients.

Our goal was to develop a test framework for SOAP Analysis in Dynamic Environments (SPADE) with a configurable transport layer and compression module that can be used to evaluate the performance of different transport protocols used with Web services under constrained network conditions. In this paper we describe the framework.

The remainder of this paper is organized as follows: Section 2 discusses related work. Our SPADE framework is presented in Section 3, and an example of its use is given in Section 4. Section 5 concludes the paper.

2 RELATED WORK

Werner et al. (Werner et al., 2005) point out that bandwidth and latency are issues when using Web services. One of the main reasons for latency can be attributed to the transport protocol. Thus, in this paper we focus on transport protocols that we consider to be currently relevant and implement support for these in our framework. However, the framework is modular so implementing support for additional protocols is possible.

In addition to the basic protocol replacement functionality, our framework also has a compression module. Since XML compression is recommended (see e.g., (Berger et al., 2003) and (Gehlen and Bergs, 2004)) when utilizing Web services in wireless networks, adding compression to the test framework allows for additional test scenarios and experiments. Our current implementation supports one type of compression, GZIP, but it is possible to extend the compression module with support for further compression mechanisms.

Avlesen M., Spjelkavik S., Vik B., Johnsen F. and Bloebaum T..
SPADE - A Test Framework for SOAP Analysis in Dynamic Environments.
DOI: 10.5220/0004351101290132
In *Proceedings of the 9th International Conference on Web Information Systems and Technologies* (WEBIST-2013), pages 129-132
ISBN: 978-989-8565-54-9
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

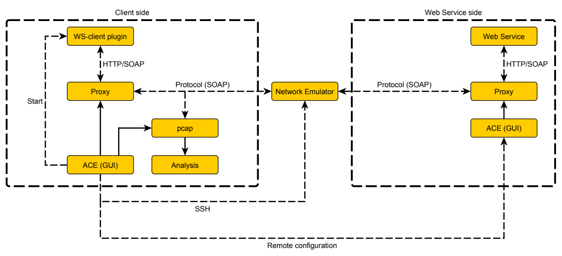

Figure 1: SPADE framework overview.

3 TEST FRAMEWORK: DESIGN AND IMPLEMENTATION

The SPADE framework was implemented in Java and consists of the following components: A Web service, a Web service client plugin, a request generator, a proxy server supporting compression and different transport protocols, a network emulator, a set of capture and analysis tools used to capture network traffic, and finally a Graphical User Interface (GUI) for generating requests and controlling the components. See Figure 1 for a graphical representation of the SPADE framework on a system level. The components are described in detail below.

3.1 The Web Service

The Web service is hosted in a GlassFish application server. The main purpose of this Web service is to accept requests and generate responses to accommodate our need for network traffic when testing the different transport protocols. The Web service generates variable payload data, dependent on what the client requests. GlassFish communicates with the proxy on the server side using SOAP over HTTP/TCP.

3.2 The Web Service Client Plugin

The Web service client used in our framework is a simple client that contacts the Web service requesting a message of a given payload size. The client has been implemented as a plugin using the Java Simple Plugin Framework. This makes it easy to extend the system with other Web service/client pairs in the future, and ensures the necessary separation of concerns between the Web service client and the SPADE framework. The client plugin communicates with the proxy on the client side using SOAP over HTTP/TCP.

3.3 Request Generator

The *RequestRun* module automates tests by configuring the network emulator, generating network traffic, and measuring the performance. The module starts with the setup phase, in which everything is made ready to start generating traffic. First, the network emulator is reset. The next step is to make the Web service client plugin ready by retrieving the WSDL for the Web service. The network emulator is then configured in accordance with the current test scenario. Finally, the phase is finished by starting *pcap* for packet capture.

The next phase is the request generation phase, where bursts of requests are sent using the Web service client plugin with a given interval in between. When all requests are sent the application waits until all responses are received, or times out. All of the requests sent exist as separate threads, and act as separate clients.

The final phase is the shut down phase where the test environment is made ready for the next test run. First, the application delays to make sure any remaining protocol test traffic is completed and queues are emptied. Then, it stops the pcap capture, saves the captured data to a file with a name containing the current test run scenario, the proxy it is connected to, and the current time. Finally, it resets the proxy, making it ready for another test run.

3.4 The Proxy Server

The proxy server implements a request/response model, and translates between different protocols, as well as supporting compression. An overview of this module is given in Figure 2.

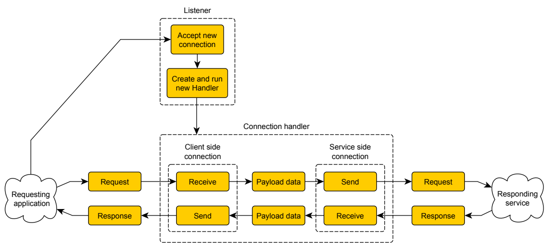

Figure 2: An overview of the message flow in the proxy server.

This module provides two connections; one client side connection that communicates with the originator of the request, and a service side connection that communicates with the Web service.

The proxy contains a compression module that can be configured to (de)compress data passing through it using GZIP (the default is no compression).

Due to the modular design it is easy to extend the framework with additional compression algorithms in the future, allowing for experiments combining compression and transport protocol optimization combinations.

The listener and connections are transport protocol specific implementations, and the client side connection and the listener communicate over the same protocol. The connections and the listener are separated as much as possible from the core proxy server, and the connection handler only passes the payload data between the two connections using send and receive commands. This structure allows for quick implementation of support for additional protocols in the future, as these implementations only need to able to separate streams and receive and send payload data over the corresponding stream. Currently, the following four protocols are supported: 1) Transmission Control Protocol (TCP) (Postel, 1981), 2) User Datagram Protocol (UDP) (Postel, 1980), 3) Stream Control Transmission Protocol (SCTP) (Stewart, 2007), and 4) Advanced Message Queuing Protocol (AMQP) (OASIS, 2012). For AMQP, we used the RabbitMQ implementation in our framework.

3.5 Network Emulator and Remote Configuration

The *netem* tool provides functionality emulating properties such as variable delay, loss, duplication, bandwidth constraints, and packet reordering. Netem is a part of Linux kernels since version 2.6, and effectively allows you to configure emulated links with specific properties using a command line interface. In our framework we installed Ubuntu 10.04 LTS on an IBX-200 (IBX, 2012) and used that as our network emulator. However, any machine that can run Linux and has two network interfaces can be used for this purpose.

In our framework, the network emulator running on the IBX-200 node is remote controlled via SSH. This is achieved by using the Java library sshj. The remote control functions like a regular SSH session, and is initiated by logging on to the IBX-200 using public key authentication. An execute command is then transferred to the IBX-200, starting the network emulator scripts including the option set for the test run.

3.6 Packet Capture and Analysis

To measure the protocol efficiency and bandwidth usage the SPADE framework captures network traffic on the client side of the system. The network traffic is captured using the Jnetpcap library to communicate with the packet capture library pcap, which is a well known open source tool used for capturing network traffic. The tool also provides basic filtering options, in order to reduce clutter and only capture the traffic type we are interested in.

Round-trip times (RTT) are not measured by pcap, but are calculated in the RequestRun module of SPADE. After the test run is complete, all the network traffic generated is saved to a file. The framework then parses the dump file and calculates throughput and protocol efficiency.

To make graphical representations of statistical data produced, this component uses the GRAL library to produce plots for throughput, packet sizes, time of capture and RTT.

3.7 Graphical User Interface

The GUI was made as a tool to make it faster and easier to set up and execute tests, and it includes the following functionality:

- Editing the parameters of all the different modules included in SPADE.
- Automatically setting parameters where this is possible.
- Saving and loading test configurations.
- Queuing RequestRuns and automatically running the queue.
- A client mode, where one SPADE instance can be remote controlled from another SPADE instance. Currently limited to transferring proxy configurations.
- Choosing where to place the data files captured by the pcap dumper.

The GUI is inspired by integrated developer environments and has a workspace overview where you create new components for the test environments. Here, the most important components are the test scenarios and the proxy servers. There are panels showing the information for each component, and panels that allow you to edit these components. The experiment queue containing the test scenarios waiting to be started are contained in an area at bottom. The GUI has support for saving and loading configurations in order to reduce the workload associated with configuring a large amount of experiments and repeating these multiple times. An overview of a typical setup is given in Figure 3.

Table 1: Result matrix. RTT = Round-trip time, SR = success rate, EFF = efficiency.

		UDP			**TCP**			**AMQP**			**SCTP**		
Link quality	**Payload**	**RTT**	**SR**	**EFF**	**RTT**	**SR**	**EFF**	**RTT**	**SR**	**EFF**	**RTT**	**SR**	**EFF**
GPRS, 40 kbps	1 kbyte	88600	64%	94%	88928	28%	42%	124530	1%	9%	59544	18%	16%
	10 kbyte	104177	2%	95%	–	0	65%	–	0	21%	94898	13%	41%
EGDE, 125 kbps	1 kbyte	25337	67%	94%	52791	83%	60%	51822	12%	38%	60130	98%	59%
	10 kbyte	120557	1%	95%	152377	17%	85%	83590	8%	61%	134512	17%	68%
UMTS, 214 kbps	1 kbyte	8965	58%	95%	18055	81%	55%	39329	35%	46%	27877	99%	62%
	10 kbyte	76703	1%	95%	123098	29%	87%	50163	7%	64%	132832	37%	81%
Sat.com. 1 Mbps	1 kbyte	2435	68%	94%	7912	89%	49%	63694	95%	64%	11761	100%	50%
	10 kbyte	72672	4%	96%	136412	96%	89%	109935	92%	87%	115614	96%	93%

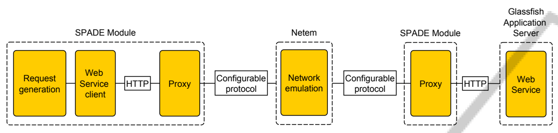

Figure 3: A typical test network setup.

4 EXAMPLE EVALUATION

Our SPADE framework was configured for experiment series using corresponding proxy protocol pairs across the emulated network. All proxies communicated with the end-point (i.e., the client or service) using HTTP, and amongst themselves using the specified transport protocols over the different emulated networks.

In total we ran 72 test runs, consisting of all the combinations of four protocols, nine test scenarios, and two different message sizes. The network traffic from these test runs was captured and saved, and then analyzed. A subset of the results generated by the framework are shown in Table 1. The complete combined test run took between six and seven hours from start to finish. This is a drawback when using emulations – everything has to run in real-time. However, since SPADE can queue defined experiment runs there is no need for human intervention during the test period.

5 CONCLUSIONS

We have discussed using compression and replacing HTTP/TCP as the transport protocol for SOAP as ways to optimize the resource usage. To support optimization experiments we have developed an easily extendable test framework, SPADE, for evaluating Web services using different transport protocols and compression algorithms. We have presented the framework, along with an example evaluation of Web services using four different transport (TCP, UDP, SCTP, and AMQP) showing how the framework can be used.

ACKNOWLEDGEMENTS

The work was funded and supervised by FFI. The framework was implemented and tested at FFI's lab facilities by students Avlesen, Spjelkavik, and Vik from HiOA. We would like to thank associate professor Hilde Hemmer at HiOA for her effort as co-supervisor during the development of this project.

REFERENCES

Berger, S., McFaddin, S., Narayanaswami, C., and Raghunath, M. (2003). Web services on mobile devices — implementation and experience. *in proceedings of the Fifth IEEE Workshop on Mobile Computing Systems and Applications*.

Gehlen, G. and Bergs, R. G. R. (2004). Performance of mobile web service access using the wireless application protocol (wap). *in Proceedings of World Wireless Congress. San Francisco, CA, USA*.

IBX (2012). Ibx-200 data sheet. http://files.ieiworld.com/ files/ news/ 080923/ IBX-200-9102G4/ P2_IBX-200-9102G4.pdf.

OASIS (2012). Oasis advanced message queuing protocol (amqp) tc. https://www.oasis-open.org/committees/ tc_home.php?wg_abbrev=amqp.

Postel, J. (1980). STD 6: User Datagram Protocol.

Postel, J. (1981). STD 7: Transmission Control Protocol: DARPA Internet Program Protocol Specification.

Stewart, R. (2007). Stream Control Transmission Protocol. RFC 4960 (Proposed Standard). Updated by RFCs 6096, 6335.

Taylor, I. J. (2005). From p2p to web services and grids. ISBN1-85233-869-5, Springer-Verlag London Limited.

W3C Working Group Note (2004). Web services architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 11 February.

Werner, C., Buschmann, C., Jäcker, T., and Fischer, S. (2005). Bandwidth and latency considerations for efficient SOAP messaging. *International Journal of Web Services Research, Vol. 3, Issue 1*.