2011

On the Design of Web Services: SOAP vs. REST

Pavan Kumar Potti
University of North Florida

ON THE DESIGN OF WEB SERVICES: SOAP vs. REST

by

Pavan Kumar Potti

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computing and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

August 2011

The thesis "ON THE DESIGN OF WEB SERVICES: SOAP vs. REST" submitted by Pavan Kumar Potti in partial fulfillment of the requirements for the degree of Master of Science in Computing and Information Sciences has been

Approved by the thesis committee: Date

Signature Deleted

6/16/2011

Dr. Sanjay P. Ahuja
Thesis Advisor and Committee Chairperson

Signature Deleted

6/20/2011

Dr. Zornitza G. Prodanoff

Signature Deleted

6/16/2011

Dr. Karthikeyan Umapathy

Accepted for the School of Computing:

Signature Deleted

6/22/11

Dr. Judith L. Solano
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

Signature Deleted

6/24/11

Dr. Peter Braza
Interim Dean of the College

Accepted for the University:

Signature Deleted

8/15/11

Dr. Len Roberson
Dean of the Graduate School

ACKNOWLEDGEMENT

I wish to thank to my parents and especially my brother Praveen for their continuous support and for giving me moral support to complete my thesis. A special thank you to my mother, who understood and forgave me for not calling her for days, when I was busy with my work. I also want to thank my friends for their help and my roommate Sethuraman, who patiently listened to me and guided me, without getting angry. In addition, thank you to all the authors of blogs and forums, at a time when there were no resources available for REST.

This thesis would not have been conceptualized and completed without the direction provided by my thesis adviser, Dr. Sanjay Ahuja. I am grateful to him for his advice. His guidance and help are much appreciated. I would also like to thank Dr. Zornitza Genova Prodanoff and Dr. Karthikeyan Umapathy for agreeing to be on my thesis committee and providing tips and encouragement throughout the thesis. Working under these three distinguished UNF faculty members has been my honor and privilege. I would also like to thank Dr. Roger Eggen and Dr. Judith L. Solano for their suggestions and for attending the thesis defense. Finally, I want to thank the University Police Department for the opening doors to the graduate lab late at night and during the holidays.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The purpose of this thesis is to compare the performance characteristics of Simple Object Access Protocol (SOAP) and Representational State Transfer (REST), which are methods of supporting interactions among Web services. They differ in both context and usage; SOAP is a protocol while REST is architecture. SOAP is a well-developed protocol used in the Web industry and is standardized by the World Wide Web Consortium (W3C). REST is the outcome of Dr. Roy Thomas Fielding's 2000 PhD dissertation, "Architectural Styles and the Design of Network-Based Software Architecture." REST is gaining in popularity due to its simplicity, scalability, and architectural dependence on the World Wide Web. Major software companies, such as Google and Amazon, among others, have started using REST. The main difference between the two methods is SOAP is a tightly coupled system, whereas REST is a loosely coupled system; both have advantages and disadvantages.

We built SOAP and REST based Web services that performed the GET, POST, PUT, and DELETE functions on a database. We utilized response time and throughput metrics to compare the performance of these Web services. In comparing the two technologies, we found REST was considerably faster, compared to SOAP, because the response times of REST were better than those of SOAP. As an ancillary outcome, we found building Web services using SOAP was easier, due to considerable tool support, whereas developing Web services using REST was time consuming, as it provides no tool support.

Chapter 1

INTRODUCTION

Web services are one category of Web solutions used in industries, such as banking, shopping, and communications. Web services can be described simply as any service offered over the Web. For example, in a weather Web application, a user opens a weather Website (client) and enters data (e.g., zip code), the site server processes the data and sends the response, in this case, a weather forecast.

A Web service mainly consists of a server and a client. The server provides defined services over the Web to clients. Clients can be of various kinds and types. Depending on the way the service was designed, clients will have minimal or no code running.

Sometimes the term "Web service" is mistaken for the term "Website." One important distinction between the two is Web services do not provide any GUI (Graphical User Interface) for the user, as Websites do. A Website and Web service are entirely different entities. AWebsite is a collection of Web pages and a Web page can contain multiple Web services. For example, a single Web page can be used for listening to music, shopping, and accessing weather, when the corresponding Web services are built into the page.

Another function of Web services is to connect software programs to each other across distant points on the Internet, transporting large amounts of data more efficiently and inexpensively than ever before. Early software ran on a single desktop, using local (inbuilt) data. Later, with the advent of distributed and client-server computing, the data was shared on different computers running on different platforms. Likewise, Web services are a type of distributed computing, developed to provide services via the Internet. "Web services work at a level of abstraction similar to the Internet and are capable of bridging any operating system, hardware platform, or programming language, just as the Web is" [Aldea]. Web services can share resources, logic, and data over a network.

The World Wide Web Consortium (W3C) characterizes a Web service as a "software system designed to support interoperable machine-to-machine interaction over a network" [Booth04]. Most Websites interact with multiple Web services acting individually or in concert, depending on the business requirements. Consider a shopping Website, functionalities such as adding items to a cart, processing them, checking out, and making the payment can be developed as different Web services. On such a site, these integrated Web services communicate with each other, to provide a pleasant shopping experience for the user.

Extensible Markup Language (XML) is the most common format for sharing data between Web services. Web service applications can coordinate series of message exchanges, using standardized XML documents. Thus, a Web service can also be

characterized as a collection of XML-based standards that enable electronic communication and interaction independently of the computer platforms or specific technologies used by the communicating parties [Answers 11].

As shown in Figure 1, Web services comprise all the logic and data and provide a standard way of interfacing between systems. They present a standard way for software programs such as DBMS, and platforms like .NET, and J2EE to interface with the network. A Web service interface receives an XML message from the network environment and transforms the message into a format understood by the interfacing systems. The implementation of this logic to transform the XML message can be created using any programming language.

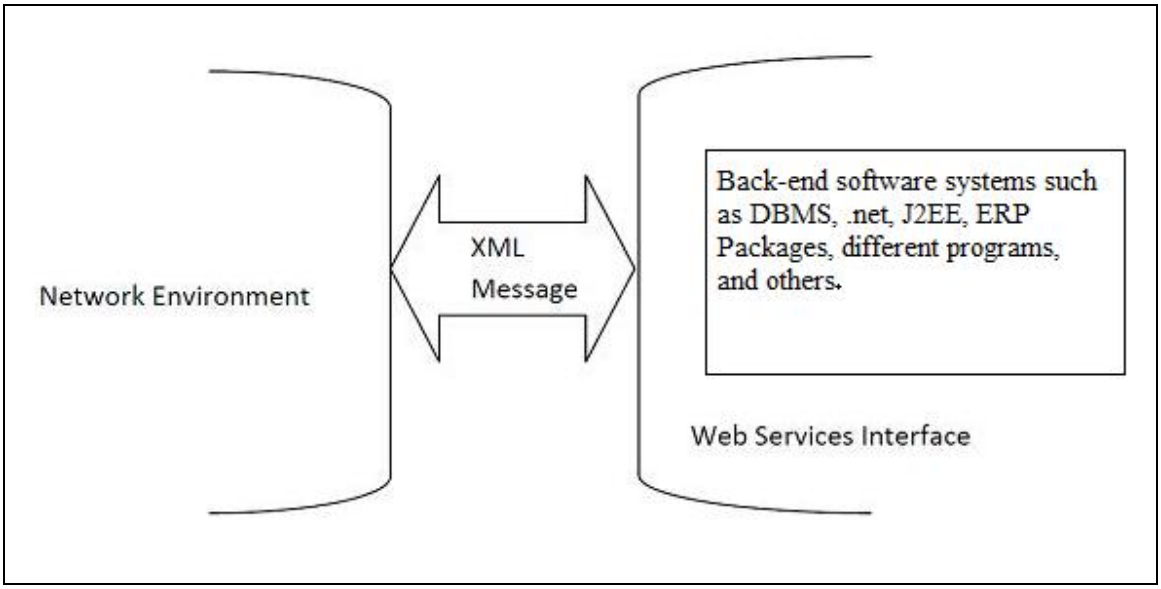

Figure 1: Web Service Interface with Back-End Systems

Distributed object technologies like Corba, Orbacus, and Java RMI were developed to meet the needs of distributed computing, but all lacked interoperability. To address this

lack of interoperability, W3C developed Simple Object Access Protocol (SOAP), which used industry standards such as XML and HTTP (Hyper Text Transfer Protocol) to facilitate exchanges between software applications, independent of the platform on which they were developed. SOAP is a technological specification developed to aid Web services interface with other systems.

Another competing approach that supports the development of Web service interfaces is Representational State Transfer (REST). REST is a set of architectural principles for designing Web applications. REST principles focus on how resource states are addressed and transferred over HTTP, by a wide range of clients written in different languages [Rodriguez08].

Although SOAP and REST are both methods of building a Web service, they differ in the way they process data and offer service. SOAP is a *protocol*, whereas REST is *architecture.* Therefore, it is not appropriate to compare these two technologies, which is a common mistake. In this study, we compared the performance of a Web service, using SOAP and REST alternately. This thesis provides a neutral assessment of the performance and services offered to developers and architects using SOAP- and REST-based methodologies while developing Web services.

Chapter 2

SOAP

SOAP, originally defined as a simple object access protocol, is a protocol for exchanging structured information in the implementation of Web services in computer networks [SOAP11]. WSDL (Web Service Description Language) is used along with SOAP to make the messages available over the Web via Web services. In a loose context, SOAP in concert with WSDL is called SOAP Web services. Without SOAP, Web services are unable to cross operating systems and platforms, and often lack interoperability. SOAP was designed to bridge this gap, making it easier for the developer to design a Web program independent of these limitations.

SOAP currently relies on HTTP as an underlying protocol, but it can use other protocols for transferring data. Microsoft was the first to start developing SOAP, and later Ariba, Commerce One, Compaq, DevelopMentor, HP, IBM, Lotus, SAP, and UserLand began contributing to its development. Early in the history of personal computing, basic processing of data consisted of one PC using simple applications that relied on local resources. Developments in networking gradually introduced concepts such as LAN (Local Area Network), MAN (Metropolitan Area Network), and WAN (Wide Area Network), allowing computers physically at different locations to communicate. Web service protocols are designed to facilitate data transfer over the Internet. Most of the Web service protocols make computers operate on similar platforms, technology, and

operating systems. The need for a protocol that was independent of these constraints gave rise to the SOAP protocol, which uses XML to communicate messages over the transport layer.

SOAP is intended to provide a minimum level of transport, on top of which more complicated interactions and protocols can be built [Newcomer02]. SOAP provides encoding rules for data and often depends on endpoint understanding. The SOAP process using WSDL is illustrated in Figure 2. A SOAP client could be designed by knowing the server endpoint URL and port address. This later became easier with the advent of WSDL and the support of IDEs (Integrated Development Environment). WSDL is an XML schema format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly and then bound to a concrete network protocol and message format, to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). "WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate" [Christensen01].

As shown in Figure 2, programs and applications at Site 1 are tied to WSDL and exposed to the Web as a Web service using SOAP. Site 1 is connected to Site 2 by SOAP. The SOAP processors at each site process the SOAP messages. Figure 2 can also be seen as a server and client communicating with each other, using the SOAP protocol. A program at Site 1 is designed using J2EE and made into a Web service using SOAP; the services

offered by the program will be exposed to the Web. The SOAP processor at Site 2 will process the messages sent by Site 1 and make the message accessible to programs or applications running at Site 2.

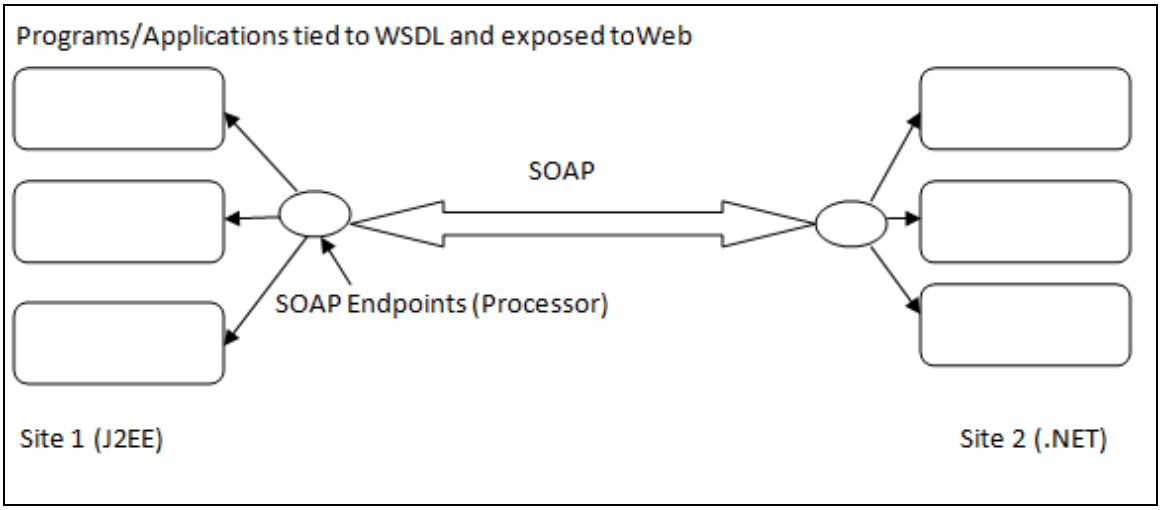

Figure 2: SOAP Mechanism

The whole process can be summarized as a client/server request response model. The client at Site 2 sends requests to the server at Site 1 as a SOAP message. The message processed at Site 1 will send the response message to Site 2 using SOAP. The SOAP message received by the client at Site 2 is processed by the SOAP processor and made available to the local programs. Thus, SOAP uses the application layer as a transportation layer. Many critics protest this is an abuse of the transport layer. SOAP was officially standardized by W3C in 2003, to be used for exposing business data using interfaces or services that exchange complete documents or map the data onto objects, using method names and input and output arguments [Newcomer02]. Following is an example of a typical SOAP message format taken from w3.org:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
 </soap:Header>
 <soap:Body>
  <m:GetStockPrice xmlns:m="http://www.example.org/stock">
   <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
 </soap:Body>
</soap:Envelope>
```

The envelope element indicates the starting and ending of the message. The header element contains optional data helpful for processing messages. The body element consists of the main XML data. In addition to being independent of HTTP, SOAP messages also have the advantage of an attachment element used for sending attachments along with the message. The attachment element is often considered best for Web services dealing with attachments. SOAP provides such a good degree of security and reliability that most banking services were designed using SOAP.

Chapter 3

REST

REST is an architecture for developing Web services. REST attempts to mimic architectures that use HTTP or similar protocols, by constraining the interface to a set of well-known standard operations (i.e., GET, PUT, POST, and DELETE for HTTP). Here, the focus is on interacting with stateful resources, rather than messages or operations. "REST architecture is designed to show how existing HTTP is enough to build a Web service and to show its scalability" [REST11].

Roy Fielding, who is also one of the principal authors of HTTP, in his doctoral thesis [Fielding00], "Architectural Styles and the Design of Network-Based Software Architectures," defined a set of architectural principles for the Web called REST. REST architectural principles can be used for designing Web service solutions. REST focuses on a system's resources, including how resource states are addressed and transferred over HTTP [Rodriguez08].

The main idea behind REST was to use well-developed HTTP for transferring data between machines, rather than using a protocol that works on top of the HTTP layer for message transfers. An application designed following REST principles would use HTTP to make calls between the machines, rather than relying on complex mechanisms like CORBA (Common Object Request Broker Architecture), RPC (Remote Procedure Call),

or SOAP. Therefore, REST applications use HTTP request functions to post data, read data, and delete data, thus using the full functionality of HTTP CRUD (Create, Read, Update, and Delete) operations. REST can also run on HTTPS, providing for the secured transmission of data.

The CRUD operations in conjunction with HTTP REST functions and relevant SQL operations are shown in Table 1.

CRUD Operations	REST Key words (HTTP)	SQL (Database) Operators
CREATE – Create or add new entries	POST	INSERT
UPDATE –Update or edit existing data	PUT	UPDATE
READ – Read, retrieve	GET	SELECT
DELETE – Delete existing data	DELETE	DELETE/DROP

Table 1: REST Operations in Parallel with CRUD and SQL Operators

Unlike SOAP, REST is not a protocol. The application of REST principles for Web services require protocols such as HTTP. Any programmer or user comfortable with HTTP should find it easy to understand and apply REST principles.

The term “RESTful” is like “object-oriented,” a language, a framework, or an application that may be designed in an object-oriented way, but that does not make its architecture the object-oriented architecture [Richardson07].

REST is an architecture for building Web services and a set of design principles analogous to a model for building a house. It is just a basic guideline for building a Web service, not a fixed set of rules.

REST sees everything as a resource, and in the words of Dr. Roy Fielding, “a resource is anything that is important enough to be referenced as a thing in itself” [Richardson07], such as a file stored in a computer, software, an attribute in a database table, or a physical object like a fruit. Resources can be static or dynamic, static like a book, or dynamic like the “weather of Jacksonville” (i.e., it always changes, but still is a resource). Static Web pages, for example, do not change if re-requested by the same or a different user. Dynamic Web pages could change and, unlike static ones, are not cacheable.

Each resource should have at least one URI (Uniform Resource Identifier) to represent it. REST refers to it as resource representation. URI is the Web address of the resource, and makes the resource available online. A resource can be accessed/referenced by its URI. Addressability is a way of representing a resource online “an addressable application exposes a URI for every piece of information it might conceivably serve” [Richardson07]. An application should be properly addressed to make it available online. A house and its address is analogous to an online resource and its URI. A house can be

located via its street address, the same way a resource should be addressed to make it available online. Consider searching for something online with a search engine. The search engine is addressable and HTTP is addressable. If the resource you are searching is addressable, the search engine will pick it up.

Statelessness is another concept of REST; statelessness means that all HTTP requests are independent of one another. Every HTTP request from the client should contain all the information for the server, so the server does not have to keep state information from client requests. Defining addressability in terms of statelessness, all possible states of the server are also resources and should be represented by a URI.

Google is one of the first Web services to have followed REST. Google Maps is completely built on the REST principle, as is Google search. In a real world example of REST using the Google search engine, searching for Florida will display some results. The results page will have an address. The address of the second results page will be different from that of the first results page. This address can be copied, stored, and accessed later. There will be no need to go to the Google Webpage to search for Florida and click next for the second page results. This is one example of a Web service following statelessness.

A stateless Web service has other advantages, such as using load-balanced servers for performance. Since no request to the Web service is dependent on another, and because the client carries all the necessary information, there is no need for coordination among

the servers. This eliminates the processing time for servers for the earlier data/process operations.

Each state of a resource is called representation. A resource can have many states and each state can be represented individually. The states of a resource are analogous to the states of matter, as in a liquid, gas, or solid. Consider the example of a Redbox (i.e., a DVD rental kiosk); we can search for the availability of a DVD at a particular kiosk online. The search will represent a DVD as available, will arrive soon, or checked out, so the same resource, the DVD, is represented in many states. This illustrates the importance of the keyword "Representation" in REST.

According to Dr. Roy Fielding [Fielding00], there are two regular perspectives on the process of architectural design; one is to start building everything without any plan, or building from scratch, and later add components to match the requirements. The other is to start building with a plan to implement all requirements. Additional features may be added later making them in accord with existing features and completing them depending on the requirements. "REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resources" [Fielding00].

The architecture of REST at the server side is as follows:

URL: Mandatory field to access Web services running at the server.

GET: All the methods of getting data from the server; the formats and interfaces the server supports for accessing the client details.

POST: All the methods of adding details to the server; all the different interfaces and formats the server supports for adding data to its database.

PUT: All the methods for updating the data at the server; different types of interfaces and formats the server supports for adding data to the database.

DELETE: All the methods for deleting the data at the server; different types of interfaces and formats the server supports for deleting data in the database.

Given the ease of design and flexibility in coding provided by REST, it has gradually become popular. Yahoo and eBay were the first ones to use REST for designing their Web services. They were later joined by popular firms such as Amazon and Google.

Chapter 4

LITERATURE REVIEW

The doctoral thesis of Roy Thomas Fielding, "Architectural Styles and Design of Network-Based Software Architectures," is where REST has its origin [Fielding00]. Dr. Fielding studied the evaluation of the architectural design of network based application software through the principled use of architectural constraints and defined a novel framework of software architecture, demonstrating how styles can be used to guide the architectural design of network-based application software. Fielding is one of the principal authors of the HTTP specification and World Wide Web (WWW) design and his architectural model REST embodies all the major principles of the WWW. In Fielding's words, "the motivation for developing REST was to create an architectural model for how the Web should work, such that it could serve as the guiding framework for the Web Protocol standards" [Fielding00].

Fielding's dissertation is the first reference available for REST, as the term "REST" was coined in the dissertation. We used this dissertation, especially chapters 5 and 6, to better understand the principles of REST. Chapter 5 talks about REST architectural style for distributed hypermedia systems. Additionally, it describes the software engineering principles guiding REST and the interaction constraints chosen to retain those principles, while contrasting them to the constraints of other architectural styles. Chapter 6 talks about how Fielding applied the standards from HTTP and WWW to REST.

Authors Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng, in "QoS Computation and Policing in Dynamic Web Service Selection," presented an open, fair, and dynamic QoS (Quality of Service) computation model for both service requesters and service providers, useful in Web services selection by implementation of and experimentation with a QoS registry in a market place application [Liu04]. Their experiments included investigating the relationship between QoS value and the business criteria, and the effectiveness of service sensitivity factors. The authors' work can be best understood as a way of showing how business related criteria can be measured and included in QoS computation. Performance speed, multiple users, and cost are some of the factors the authors took into consideration while designing their experiments.

Authors Ludmila Cherkasova et al., in [Cherkasova03] "Measuring and Characterizing End-to-End Internet Service Performance," measured various parameters to study Web service performance. The authors developed a model EtE (end-to-end) monitor to measure Web site performance. EtE system logic seems simple, but it was still very useful in measuring various parameters. This model passively collects packet traces from the server site to determine service performance characteristics. The authors effectively illustrated the results of their work with the use of graphs for three case studies of three different Web services. In one of the case studies, which focused on response time and its factors, the authors discussed such factors as duration, size, and execution time in measuring response time for large file sizes.

The paper titled “Developed Web Services Choreography Standards - The Case of REST vs. SOAP” [Muehlen05], by Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson, is a very good reference paper comparing the two approaches to developing Web services, i.e., REST and SOAP. One main difference stressed in this paper and elsewhere is that SOAP is a tightly coupled design similar to RPC (Remote Procedure Call), and REST is a loosely coupled design similar to navigating Web links. As a study of two opposing standards based on SOAP and REST, this paper explains the advantages and disadvantages of REST-based and SOAP-based approaches to process integration. SOAP has the advantage of tight coupling of operations, while REST has the advantage of scalability and lightweight access to its operations. REST has the disadvantage of managing the name space with a large number of objects, and a disadvantage of SOAP is the need for dedicated client ports for different types of notification. This paper also describes how REST principles played a role in standardizing SOAP 1.2. This is also one of the first papers calling attention to the fact that SOAP and REST are incommensurable.

Chapter 5

RESEARCH METHODOLOGY

The main idea of this study was to compare the performance of Web services using the two major Web service building technologies SOAP and REST. Both technologies use HTTP as a layer of transport. SOAP is a protocol used for building Web services, whereas, REST is an architecture for building Web services. XML is the medium of transporting messages from server to client for SOAP. We developed the REST Web service first with string arrays and byte arrays for data transfer, but finally used XML instead to make a fair comparison between the technologies.

REST logic basically runs on four different HTTP functions: GET (to retrieve data from the server), POST (to post data at the server), PUT (for updating data), and DELETE (for deleting data). Every Web service runs primarily on these four functions, though services may use variants or combinations of these. SOAP, similar to REST, also depends upon HTTP functions. For the purpose of this thesis, we developed SOAP-based and REST-based Web services that had GET, POST, PUT, and DELETE functionalities. We compared the performances of REST and SOAP when using these functions.

In this thesis, we utilized a customer class, with the 10 different fields listed below.

First name
Last name
SSN
ID (primary key)

Salary
Email
Active status
Mobile
City
Country

5.1 Functions Supported by SOAP-Based and REST-Based Web Services

The basic methodology used in the implementation of the four functions for this thesis was the same for the SOAP and the REST Web services.

5.1.1 GET Function

User data stored in a database (Oracle 10g) at the server can be retrieved and sent to the client. The client can request a particular customer's details by sending the customer ID. The server then processes the ID with the database (Oracle) using jdbc (java database connectivity), retrieves the information, and sends it to the client in the form of a string. The customer details received from the server will be displayed at the client.

5.1.2 POST Function

This function was programmed as a simple definition for posting information at the server. The client sends the data in XML format to the server with all the information. The server processes the XML, stores the data in temporary variables, and then stores it in the database.

5.1.3 PUT Function

The PUT function is used to update data already existing at the server. The client sends data to be updated to the server in the form of XML data with the customer ID. The server receives the XML, processes it, and updates the customer's data in the database based on the customer ID received.

5.1.4 DELETE Function

The DELETE function is used to delete the whole record of a customer from the database. The client will send the customer ID of the customer be deleted to the server. The server processes the data and deletes the customer record from the database.

To gain a more thorough understanding of the effect of the two technologies on performance, studies were completed for both wired and wireless connectivity. The same code and same functions were tested on both wire-connected PCs (10/100 Ethernet connected) and on two wireless-connected PCs. The main server was the same for both types of connectivity.

5.2 The Metrics and the Functions

The basic functionality for SOAP and REST were the same in that the client requested a file from the server and the server sent the file to the client. For SOAP, we created a customer class at the server side with the variables listed earlier. The client input was provided by creating an instance of a Web service class and manually entering the details of the customer.

The code at the server supported all the primary functions differently by methods appropriate for SOAP and REST. Now that we have looked at the variables of XML and the basic functionality of the four functions, let us examine the functions, and how the metrics were measured.

5.2.1 Response Time

The general scenario for generating response times involved the client sending some data to the server at an instance of time A (measured in milliseconds). The server received the request, processed it, and sent the response to the client. The client received the response completely at some time B (measured in milliseconds). The response time was measured as the difference between times A and B. The concept of multithreading was used to simulate multiple clients accessing the server at a given time. Each request sent from a client was a thread and each thread had its own number. The concept of synchronization with a correlation coefficient was also used, so each thread would be initialized in order. The threads were divided equally between the two clients and ran at the same time. A run of 20 clients (threads) ran 10 threads on each machine. The response time was calculated for each thread separately. For every run, the arithmetic mean of the thread response times was measured and considered as the response time for that run. For a run of 10 clients, each thread response time was measured. The arithmetic mean of the response times was calculated and considered to be the response time for 10 clients (5 threads running on each client).

5.2.1.1 Response Time and the GET Function

The methodology used to measure response time relative to each of the functions was the same for both the REST and SOAP implementations. The GET function is used to get customer details from the server. Customer details were stored in a table (Customer table) of an Oracle 10g database. The client sent a request to the server, by sending a customer ID (primary key of the customer table). Upon receiving the request, the server stored the customer ID in a temporary variable and connected to the database using a driver. Later the database was searched for the customer ID. Upon a match, the customer details were stored in temporary string variables, in turn, all strings were stored in a string array. The string array was sent to the client, to be stored locally for further processing.

Following is the skeleton of the code for measuring response time of the GET function:

```
Client Implementation Thread class
{
Thread run method
Public void run ( )
{
GETmethod (with correlation ID);
}
GETmethod ()
{
System time in milliseconds A of a particular thread X;
Code for GET Operation;
System time in milliseconds B of a particular thread X;
}
}
```

Using multithreading with synchronization and a correlation coefficient, each thread had a different ID. To make sure each thread requested different customer details, rather than

all threads requesting the same customer details, each thread requested the customer ID based on the thread ID. Response times were taken with threads ranging from 1 to 40, for both REST and SOAP (wired and wireless).

5.2.1.2 Response Time and the POST Function

POST is used to add data at the server. The client sent customer data in XML format to the server. The server processed the received XML and stored the customer data in local variables. The server connected to the database through a driver and stored the data in the Customer table of an Oracle 10g database. The server then sent a response to the client.

Following is the skeleton of the code for measuring response time of the POST function:

```
Client Implementation Thread class
{
Thread run method
Public void run ( )
{
POSTmethod (with correlation ID);
}
POSTmethod ()
{
System time in milliseconds A, of a particular thread X;

Code for POST Operation;
System time in milliseconds B, of a particular thread X;
}
}
```

Using multithreading with synchronization and a correlation coefficient, each thread had a different ID. To make sure each thread posted different data, the customer ID of every

thread was based on the thread ID. Response times were taken with threads ranging from 1 to 40 (each thread had complete customer data), for REST and SOAP (wired and wireless).

5.2.1.3 Response Time and the PUT Function

PUT is used to update data at the server. The client sent customer data to the server to be updated. The server, upon receiving the data, processed it and updated the database. The client sent data to the server in XML format with the customer ID of the customer to be updated. The server processed the received XML and stored the customer data in local variables. The server connected to the database through a driver and updated the data in the Customer table of an Oracle 10g database. The server then sent a response to the client.

Following is the skeleton of the code for measuring response time of the PUT function:

```
Client Implementation Thread class
{
Thread run method
Public void run ( )
{
PUTmethod (with correlation ID);
}
PUTmethod ()
{
System time in milliseconds A, of a particular thread X;
Code for Put Operation with customer ID;
System time in milliseconds B, of a particular thread X;
}
}
```

Using multithreading with synchronization and a correlation coefficient, every thread had a different ID. To make sure every thread updated different customer details, the customer ID of every thread was based on the thread ID.

5.2.1.4 Response Time and the DELETE Function

DELETE is used to delete data at the server. The client sent customer details to the server to be deleted. The server processed the data and deleted the details from the database. The DELETE function was coded based on the customer ID (the primary key of the Customer table). The client sent a customer ID to be deleted in the form of a string. Upon receiving the string, the server processed it and stored the data in local variables. The server connected to the database through a driver and deleted the data from the Customer table of an Oracle 10g database. Finally, the server sent a response to the client.

Following is the skeleton of the code for measuring the response time of the DELETE function:

```
Client Implementation Thread class
{
Thread run method
Public void run ( )
{
DELETEmethod (with correlation ID);
}
DELETEmethod ()
{
System time in milliseconds A, of a particular thread X;
Code for DELETE Operation with customer ID;
System time in milliseconds B, of a particular thread X;
}
}
```

Using multithreading with synchronization and a correlation coefficient, each thread had a different ID. To make sure each thread updated different customer details, the customer ID of every thread was based on the thread ID. Response times were not measured for the DELETE function individually, but were combined with all four functions and measured as explained in the next section.

5.2.1.5 Response Time and All Four Functions

The previous sections talked about the individual functionality of the functions of GET, POST, PUT, and DELETE and the measurement of response times in a wired and wireless environment. To compare REST and SOAP further, we coded a function involving all four functions together in a sequence. First, the client sent customer details in XML format to the server. The server processed the XML, connected to the database, and stored the information (POST). Second, the client issued a GET request for customer details (GET). The GET request was followed by an update. The client sent an update request to the server for the data that had been added, using the POST function. Later, the client sent a final request to the server to delete the data that had been added and updated before.

Following is the skeleton of the code for measuring response time of the four functions in sequence:

```
Client Implementation Thread class
{
Thread run method
Public void run ( )
{
```

```
POSTmethod (with correlation ID);
GETmethod (with correlation ID);
PUTmethod (with correlation ID);
DELETE method (with correlation ID);
}
POSTmethod ()
{
System time in milliseconds A, of a particular thread X; (Timer started for the
thread based on correlation ID)
Code for POST Operation with customer ID;
}
GETmethod ()
{
Code for Get Operation with customer ID;
}
PUTmethod ()
{
Code for Put Operation with customer ID;
}
DELETEmethod ()
{
Code for Delete Operation with customer ID;
System time in milliseconds A, of a particular thread X; (Timer end for the thread
based on correlation ID)
}
}
```

The skeleton code provides insight into the functionality needed for testing and measuring response time. All the functions were coded with correlation IDs and customer IDs. Every time a thread initialized, it took an ID based on the correlation ID. Later, the customer ID was based on the thread ID. First, a thread was initialized and took an ID. It started executing the POST function and sent customer details to be added to the server (customer ID was based on the thread ID). Later, the thread executed the GET function and retrieved the customer's details with the customer ID. This was followed by update (POST) and DELETE functions, based on the customer ID. The cycle involved adding customer data, getting the data added, and deleting the data added.

The starting and ending times of a thread were measured in the GET and DELETE functions (first line of code in the GET function and last line of code in the DELETE function). The difference between the starting and ending times was the execution time of the thread. Response times were taken with threads ranging from 1 to 40 (each thread had customer data), for both REST and SOAP.

5.2.2 Throughput

There are various types and definitions of throughput. Generally speaking, it is defined as the data processed in a unit of time. For this thesis, the throughput was defined as the number of bytes per second and the number of clients per second.

5.2.2.1 Throughput in KiloBytes per Second

Throughput in KiloBytes per second was calculated by dividing the file size in KiloBytes by the response time in seconds. This involved another GET function. The GET function used before was based on a simple file size, typically a few bytes of XML data. The GET function used for this metric was based on accessing files ranging in size from a few KiloBytes to 5 megabytes. The client sent a request to the server, the server processed the request, and sent a response to the client. This involved considerable study and experimentation to finalize the function, since SOAP has its own technology for sending attachments, though REST does not. Addtionally, SOAP has the advantage of providing for internal conversion of files. The same logic is used for both SOAP and REST for this function.

Files of type .png were stored in the local directory of the server. The client sent a request for a file with the filename to the server. The server processed the request, retrieved the file from the local drive, and sent the file to the server. The same logic for this function was used in both the REST and SOAP implementations.

Following is the skeleton of the code for measuring throughput:

```
Client class
{
System time in milliseconds;-
Code for requesting and getting the file, stored in temp variable;
System time in milliseconds;
}
```

Every time a request was sent for a specific file, time was measured before and after the code for getting the file. The difference between the two times was the response time. After getting response time, throughput for each file was calculated using the following formula: Throughput (bytes per second) = file size/response time in seconds. Throughput was computed and graphs were plotted for both SOAP and REST (wired and wireless).

5.2.2.2 Throughput in Clients per Second

Throughput in clients per second was calculated by dividing the number of clients by the response time in seconds. The response time and data used for the GET function were used for calculating throughput and plotting graphs for both SOAP and REST (wired and wireless). The following formula was used for calculating throughput: Throughput (clients per second) = number of clients/response time in seconds.

5.3 Experimental Environment

Figure 3 illustrates the experimental environment used in this thesis.

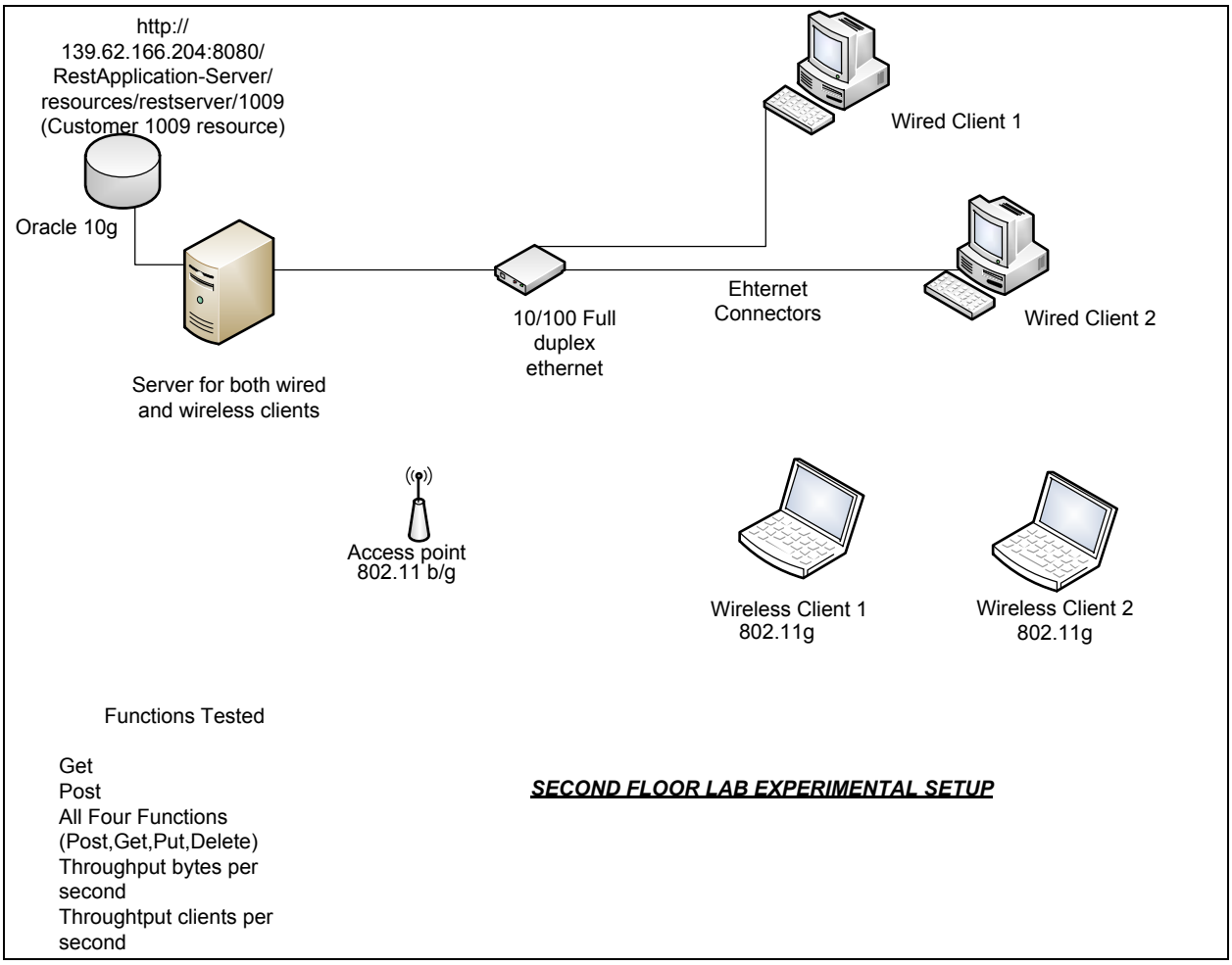

Figure 3: Experimental Setup

Both wired and wireless clients are shown in Figure 3. The same server was used for both wired and wireless clients. The customer table runs on an Oracle 10g database at the server, and the server and wired clients are connected by a 10/100 Full duplex Ethernet modem. Wireless clients are connected to the server from the same lab through an access point using 802.11g protocol. This figure also lists the experiments conducted

for the thesis, response time for GET, POST, and "All Four Functions," as well as throughput in KiloBytes per second and throughput in clients per second.

The server and two wired clients were identical systems both in hardware and in software and were connected by an Ethernet modem. The details of the configuration were as follows:

Processor: Intel ® Pentium ® 4 CPU 3.00 GHZ 2.99 GHz
RAM: 0.99 GB
Operating System: Microsoft XP Professional Version 2002 Service Pack 3
Hard Drive: Maxtor 6Y080M0
Network Adapter: Broadcom NetXtreme 57xx Gigabit Controller
Connectivity: All three systems are connected to 10/100 Full duplex Ethernet

The wireless clients differed in configuration, so the details are shown separately.

Wireless client 1:

Processor: Intel ® Dual Core CPU T2050 @ 1.73 GHz
RAM: 1.99 GB
Operating System: Microsoft XP Professional Version 2002 Service Pack 3
Hard Drive: Hitachi HTS541060G9SA00
Network Adapter: Dell Wireless 1390 mini-card
Connectivity: Connected to the server wirelessly from second floor lab using 802.11g.

Wireless client 2:

Processor: Intel ® Core ™ i5 CPU M430 @ 2.27 GHZ 2.27 GHz
RAM: 4.00 GB
Operating System: Microsoft XP Professional Version 2002 Service Pack 3
Hard Drive: ST9320325AS
Network Adapter: Atheros AR5B93 Wireless Network Adapter
Connectivity: Connected to the server wirelessly from second floor lab using 802.11g.

Software on the server side:

Netbeans 6.7
Application server: Glass Fish 3 Prelude, Glass Fish 3
Java: JDK 1.6
SOAP: JAX-WS 2.0
REST: JAX-RS 1.1
Oracle 10g

Software on the client side:

Netbeans 6.7
Java: JDK 1.6

Netbeans 6.7 IDE was used at both server and client machines for developing and accessing Web services. Oracle 10g was used at the server side for the customer database.

Chapter 6

RESULTS

6.1 The Metrics and the Functions

Libraries for REST were still at the development stage, so no support was available. We used multithreading to simulate multiple clients and were not able to solve the run time errors for the PUT and DELETE functions. In order to include them in our study, we used them in all four functions and took care of the errors by operating the functions in sequence.

Graphs were plotted for the functions GET, PUT, and all four functions, as well as throughputs for SOAP and REST (both wired and wireless networks). The network speed factor played a significant role in the wired and wireless networks, so we grouped the graphs separately for wired and wireless results.

Wireless clients used for this thesis were of a new configuration (dual and virtual cores) compared to those in the wired network. Well-developed support for SOAP took advantage of this new configuration. Graphs are arranged by function with individual graphs (wired first followed by wireless) followed by comparison graphs and finally the throughput graphs.

6.1.1 Response Time and the GET Function

The GET function enables a client to send a customer ID to the server. The server gets the details from the database and sends the response to the client. Multithreading was used to depict multiple clients accessing the server at the same time. As shown in Figures 4 through 9, the x-axis represents the number of simultaneous service requests (threads) and the y-axis represents the response time in milliseconds.

6.1.1.1 Wired Network

Figures 4 and 5 plot the response time of REST and SOAP when dealing with differing numbers of simultaneous service requests in a wired environment.

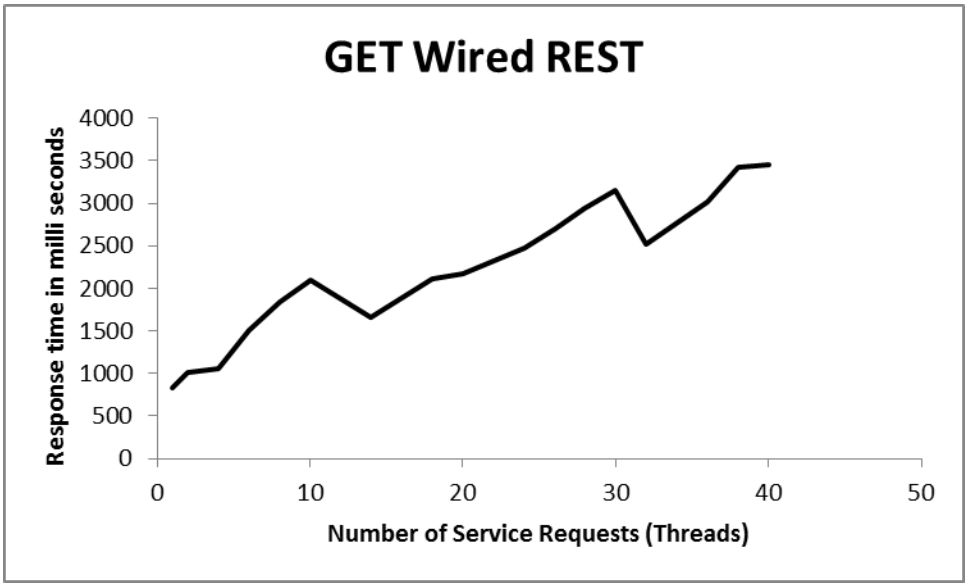

Figure 4: GET Wired REST

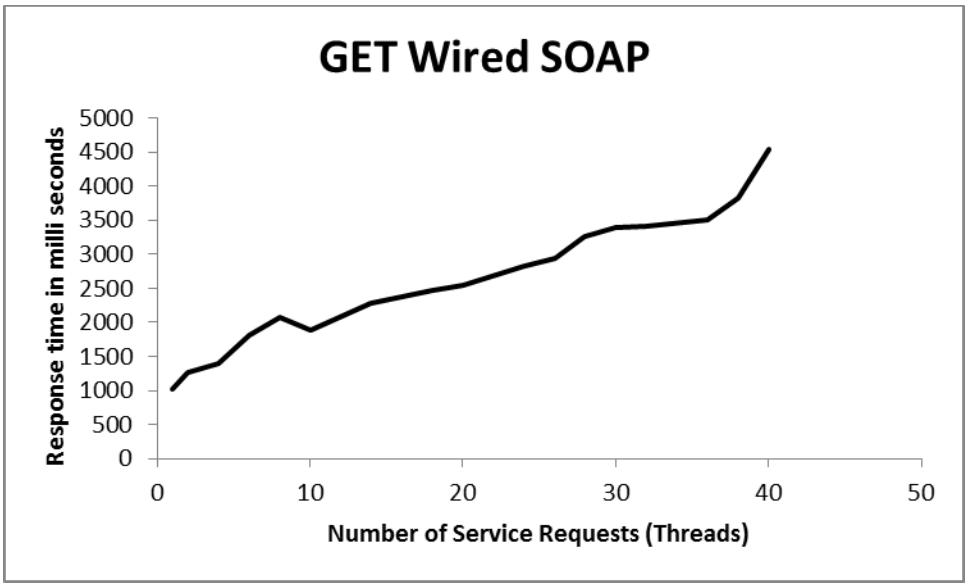

Figure 5: GET Wired SOAP

6.1.1.2 Wireless Network

Figures 6 and 7 plot the response time of REST and SOAP when dealing with differing numbers of simultaneous service requests in a wireless environment.

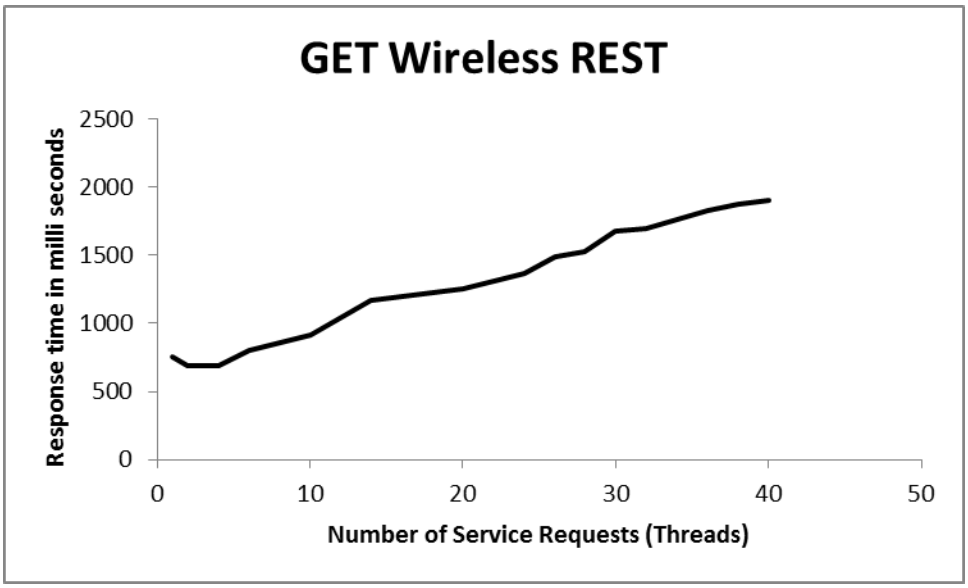

Figure 6: GET Wireless REST

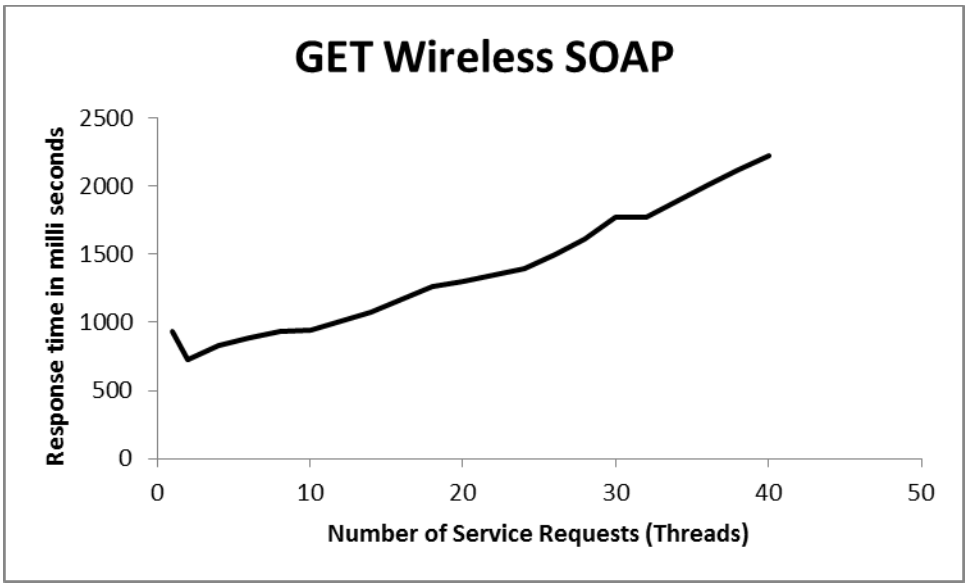

Figure 7: GET Wireless SOAP

6.1.1.3 REST vs. SOAP Comparison Graphs

Figures 8 and 9 show the comparison of response time for REST vs. SOAP for the GET function in a wired and then a wireless environment.

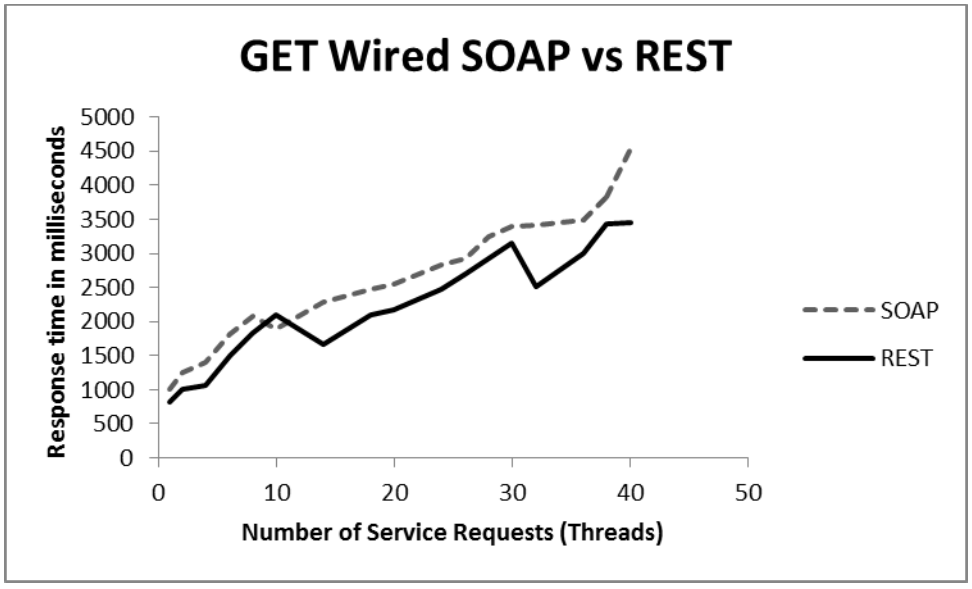

Figure 8: GET Wired SOAP vs. REST

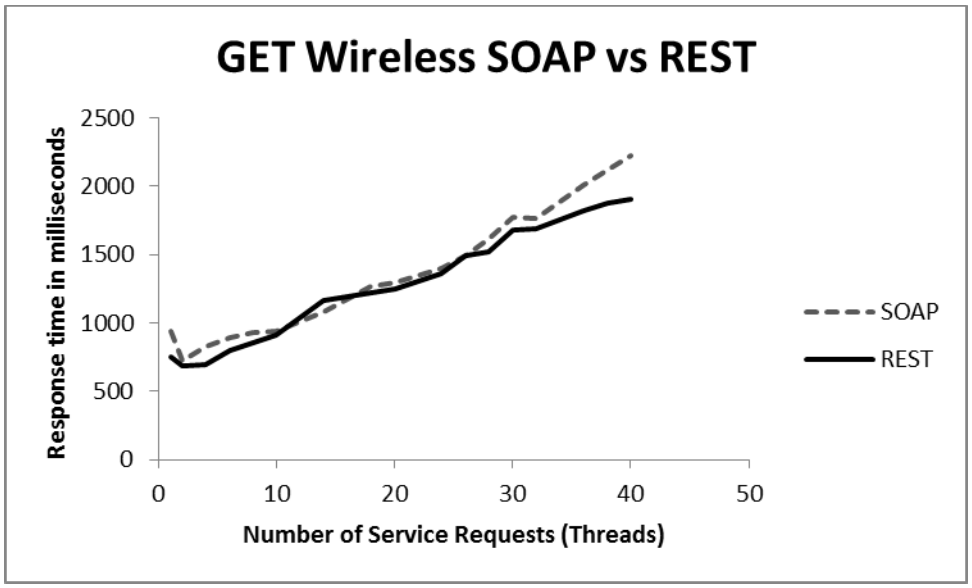

Figure 9: GET Wireless SOAP vs. REST

GET is the most used function in Web services. A lot of thin clients and browsers send requests for information to the server. As shown in Figures 8 and 9, REST and SOAP response times were almost the same (more in wireless because of the system configuration). The more highly developed SOAP had the advantage of expected data, but REST was competitive. As shown in the graphs, REST response times were comparatively better as the number of clients increased.

6.1.2 Response Time and the POST Function

POST enables a client to send data (with different customer IDs) to be added to the server. The server processes the data, stores the data in the database and sends a response to the client. As shown in Figures 10 through 15, the x-axis represents the number of simultaneous clients, and the y-axis represents the response time measured in milliseconds.

6.1.2.1 Wired Network

Figures 10 and 11 plot the response times of REST and SOAP when dealing with differing numbers of simultaneous service requests in a wired environment.

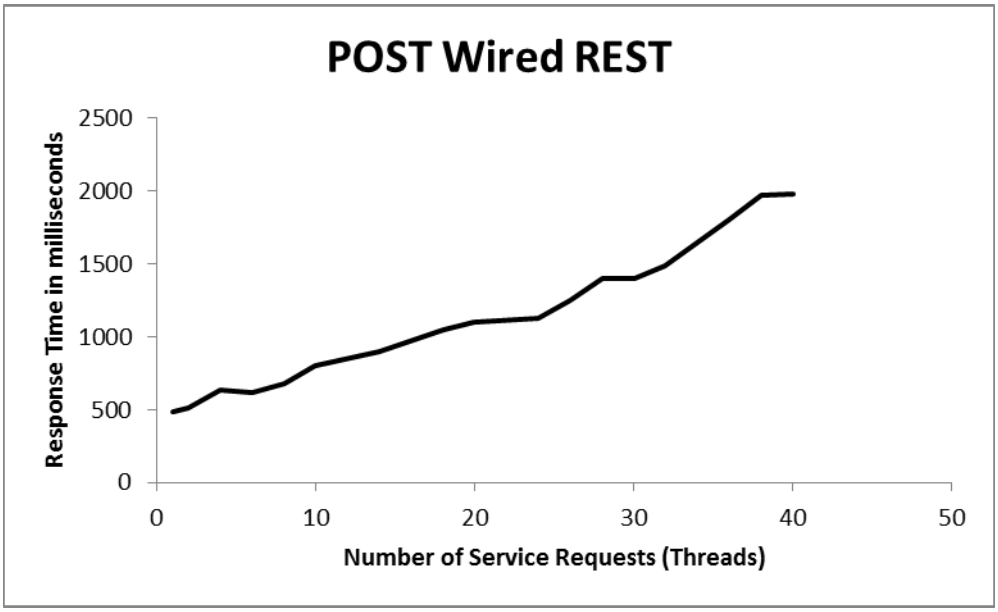

Figure 10: POST Wired REST

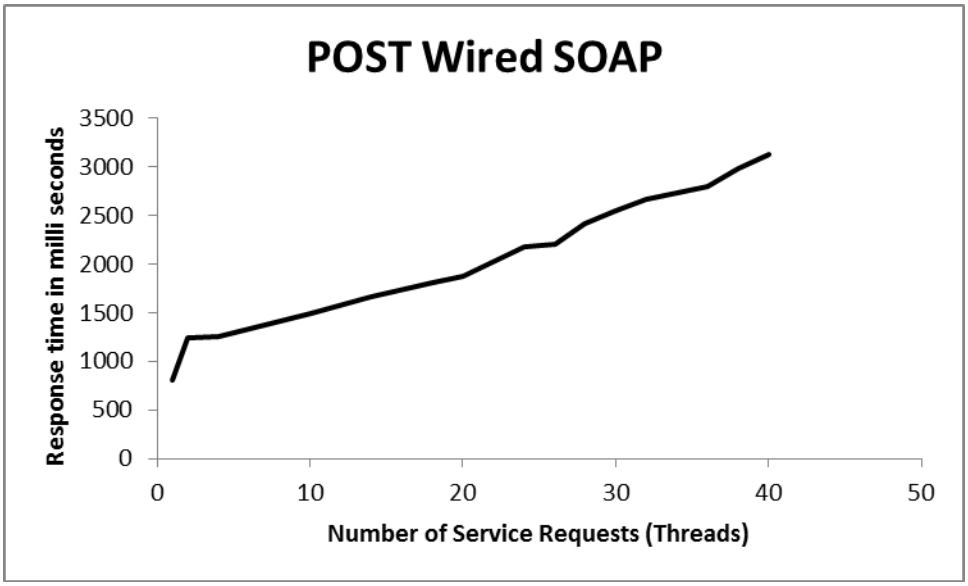

Figure 11: POST Wired SOAP

6.1.2.2 Wireless Network

Figures 12 and 13 plot the response times of REST and SOAP when dealing with differing numbers of simultaneous service requests in a wireless environment.

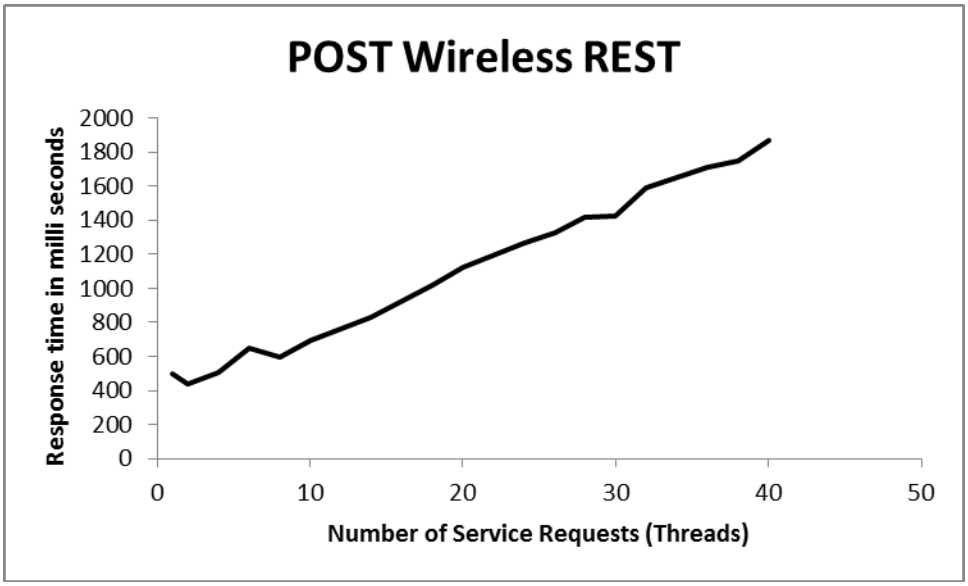

Figure 12: POST Wireless REST

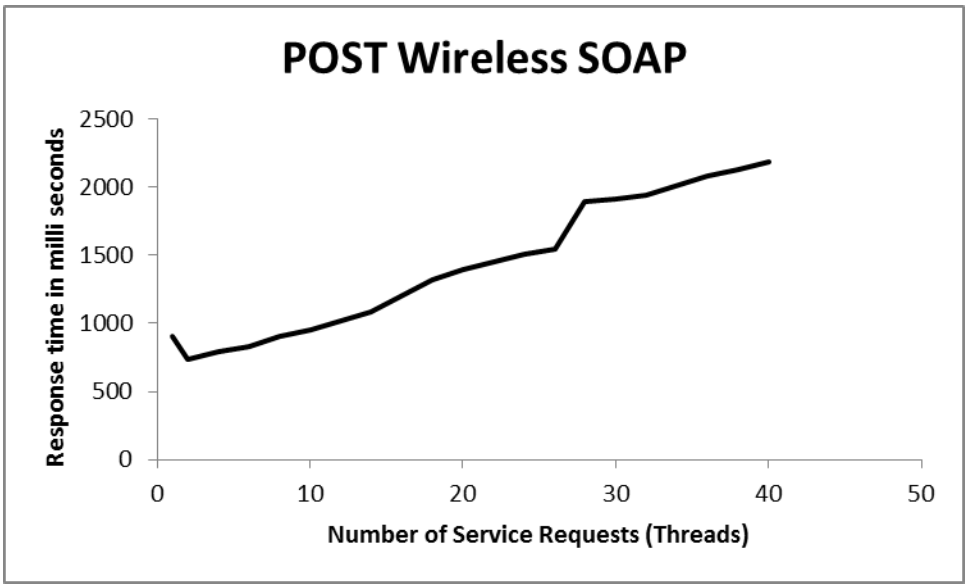

Figure 13: POST Wireless SOAP

6.1.2.3 REST vs. SOAP Comparison Graphs

Figures 14 and 15 show the comparison of response times for REST vs. SOAP for the POST function in a wired and then a wireless environment.

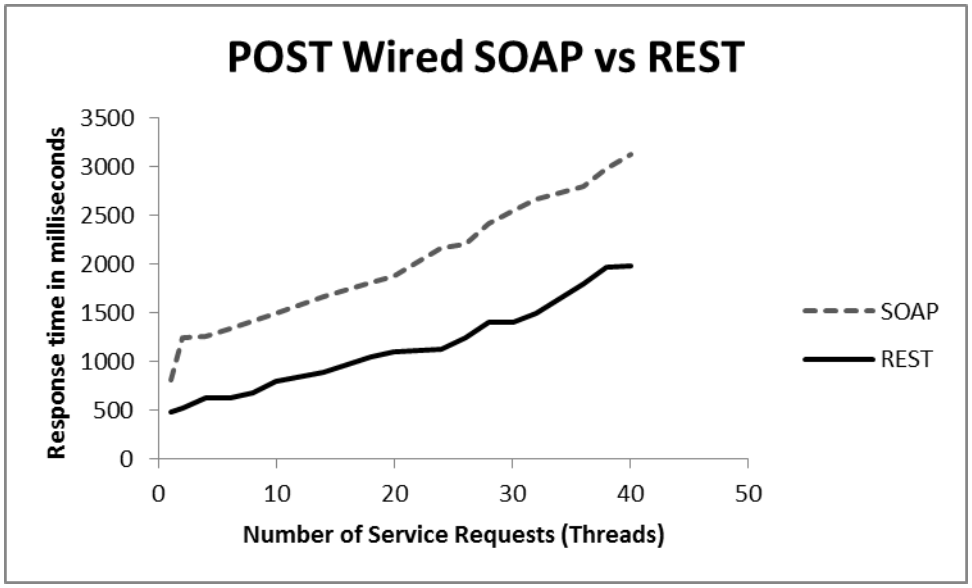

Figure 14: POST Wired SOAP vs. REST

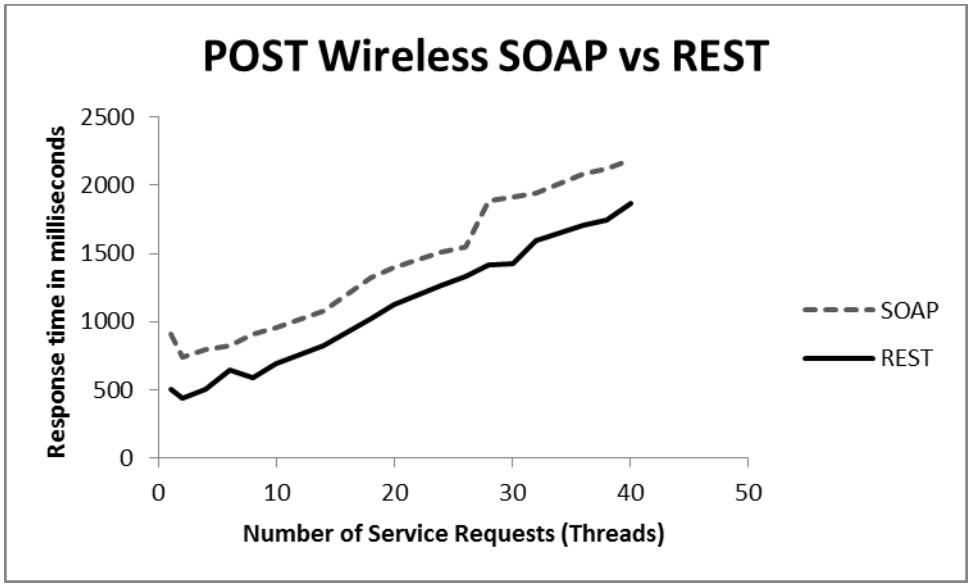

Figure 15: POST Wireless SOAP vs. REST

Figures 14 and 15 show REST was comparatively better than SOAP and the response times were better, as the number of simultaneous clients increased. Adding new data required some pre-checking and conversion effort for SOAP and often took a considerable amount of time; thus, the performance factor difference between SOAP and REST.

6.1.3 Response Time and All Four Functions

This was a unique function developed to cover all the functions supported by REST. The function contained a series of functions executed in order one after the other: PUT, GET, POST, and DELETE.

A thread carried customer details (with a unique customer ID) to the server. It later requested the details of the added customer, updated the customer details, and sent a delete request to delete the customer. Figures 16 through 21 show the response time in milliseconds on the y-axis and number of simultaneous clients on the x-axis.

6.1.3.1 Wired Network

Figures 16 and 17 plot the response times of REST and SOAP when dealing with differing numbers of simultaneous service requests in a wired environment.

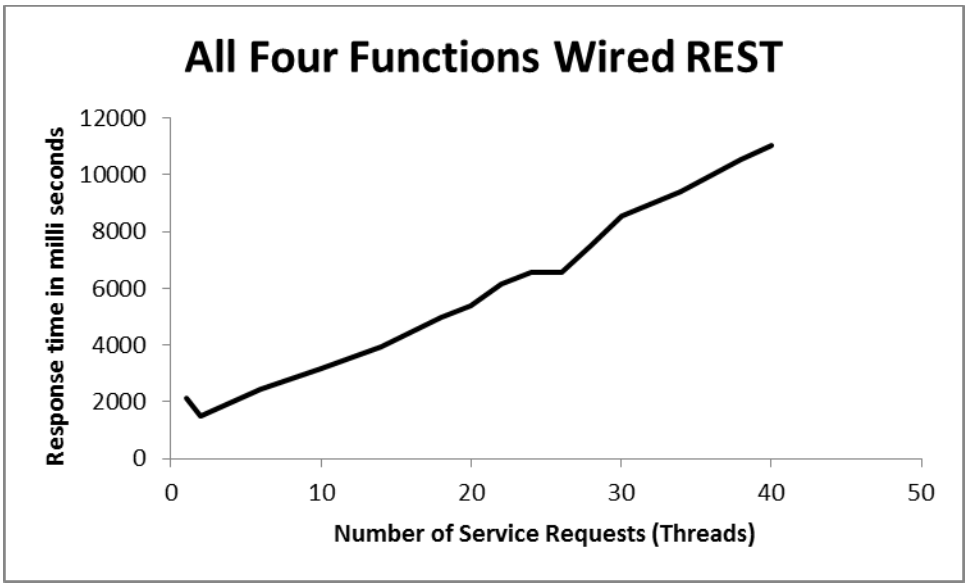

Figure 16: All Four Functions Wired REST

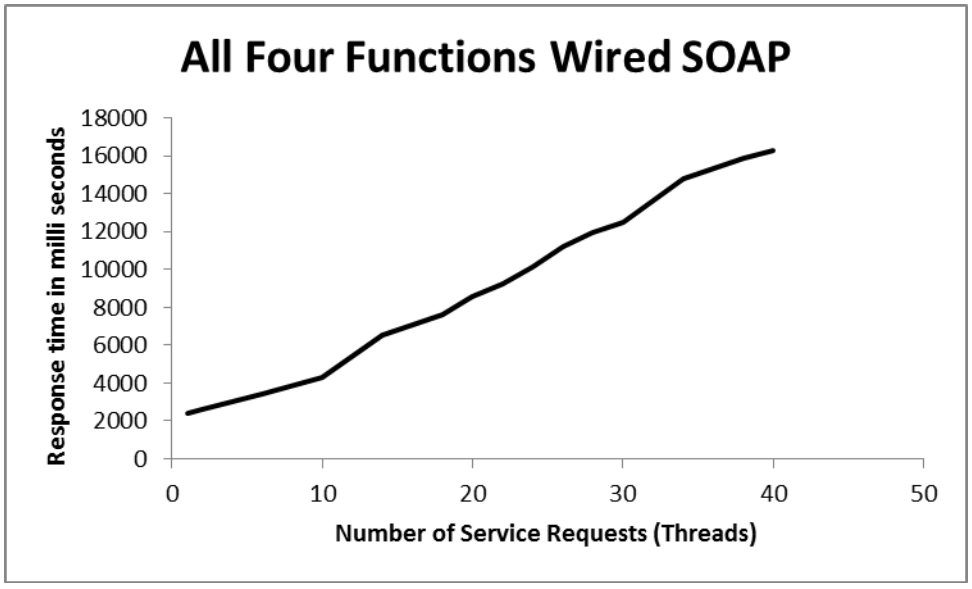

Figure 17: All Four Functions Wired SOAP

6.1.3.2 Wireless Network

Figures 18 and 19 plot the response times of REST and SOAP when dealing with differing numbers of simultaneous service requests in a wireless environment.

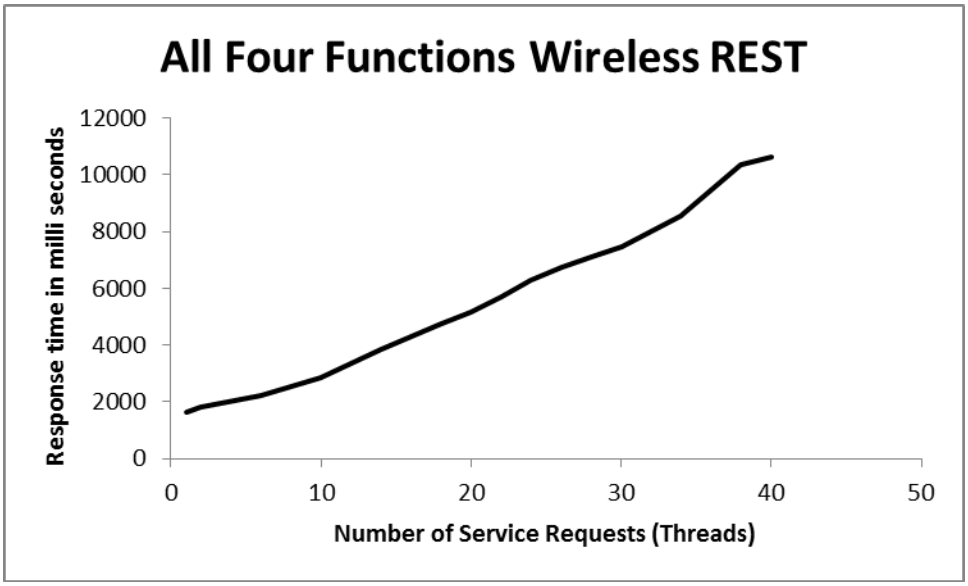

Figure 18: All Four Functions Wireless REST

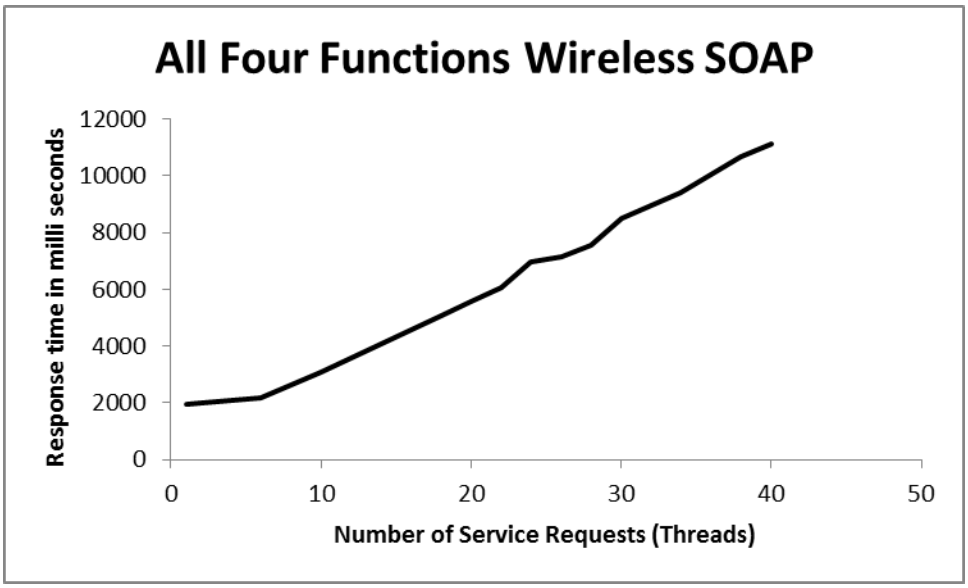

Figure 19: All Four Functions Wireless SOAP

6.1.3.3 REST vs. SOAP Comparison Graphs

Figures 20 and 21 show the comparison for REST vs. SOAP for all four functions in a wired and then a wireless environment.

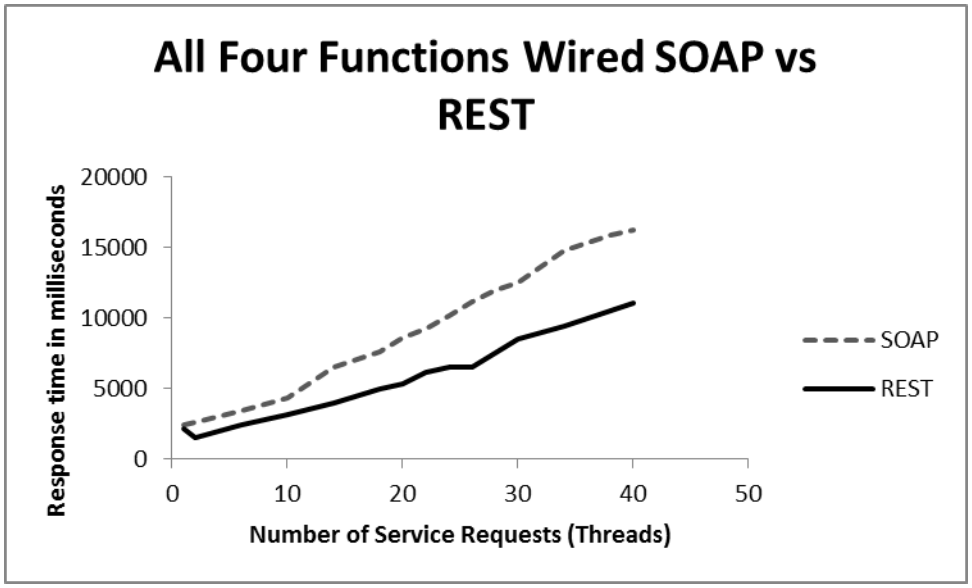

Figure 20: All Four Functions Wired SOAP vs. REST

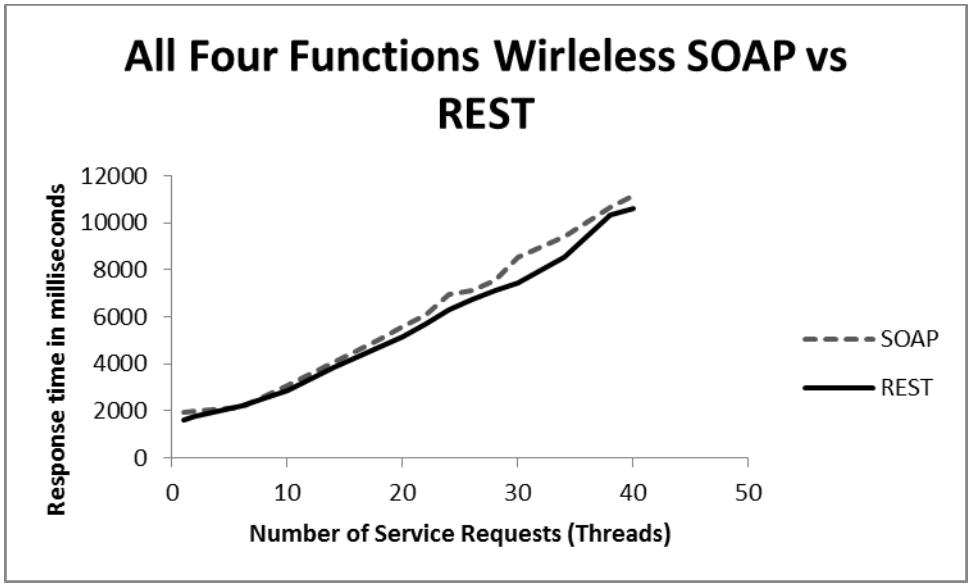

Figure 21: All Four Functions Wireless SOAP vs. REST

In Figures 20 and 21, REST appears to have performed comparatively better than SOAP, for all four functions. Among the four different functions, the GET function constitutes the majority of the response time, affecting the overall functionality and response time, accounting for the major performance difference.

6.1.4 Throughput

Throughput is defined as the average rate of successful data transmission over a channel. Files of sizes ranging from 76 KiloBytes to 5083 KiloBytes were used to compute the throughput in bytes. The number of simultaneous clients used to compute throughput ranged from 0 to 40. The sections below first compare the response time between SOAP and REST for bigger file sizes and large numbers of clients. Later sections compare the throughput metric calculated from the response times.

6.1.4.1 Response Time by File Size

Figures 22 through 25 are the graphs for response times of different file sizes measured from the instant of time the client requested a file to the instant of time the file was completely received from the server. This is another GET function with a single client requesting the data of larger file sizes from the server. The x-axis represents the file size and the y-axis represents the response time in milliseconds.

6.1.4.1.1 Wired Network

Figures 22 and 23 plot the response times of REST and SOAP when dealing with differing file sizes in a wired environment.

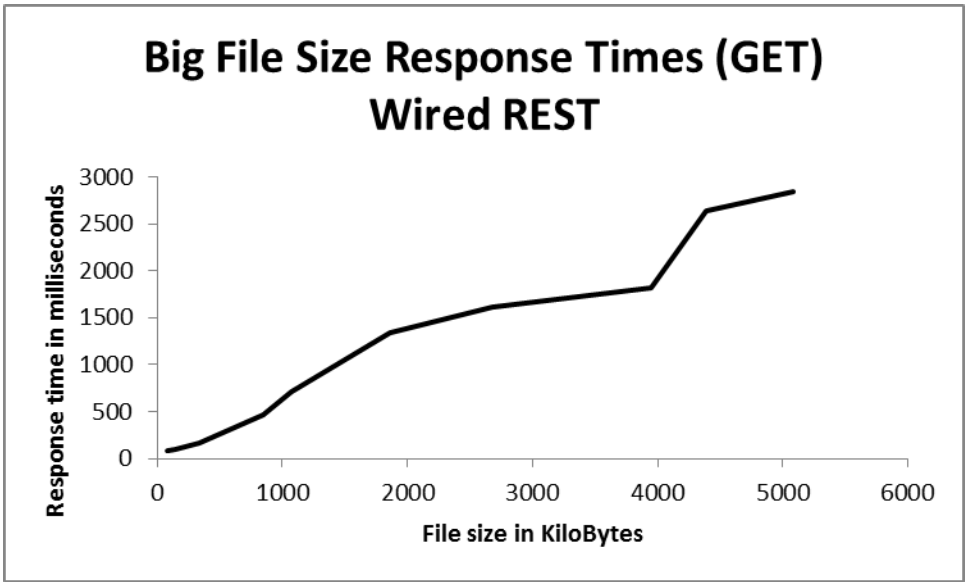

Figure 22: Big File Size Response Times (GET) Wired REST

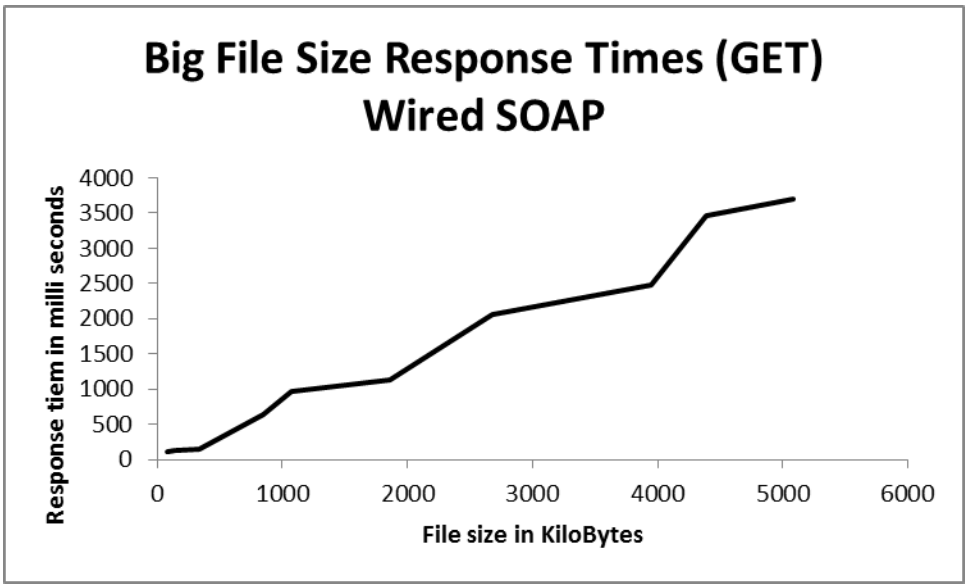

Figure 23: Big File Size Response Times (GET) Wired SOAP

6.1.4.1.2 Wireless Network

Figures 24 and 25 plot the response times of REST and SOAP when dealing with differing file sizes in a wireless environment.

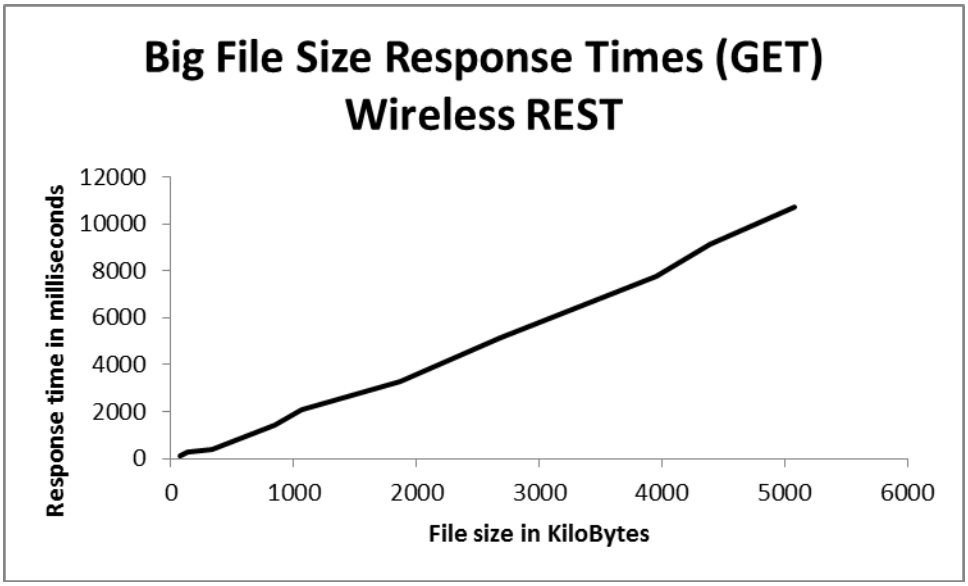

Figure 24: Big File Size Response Times (GET) Wireless REST

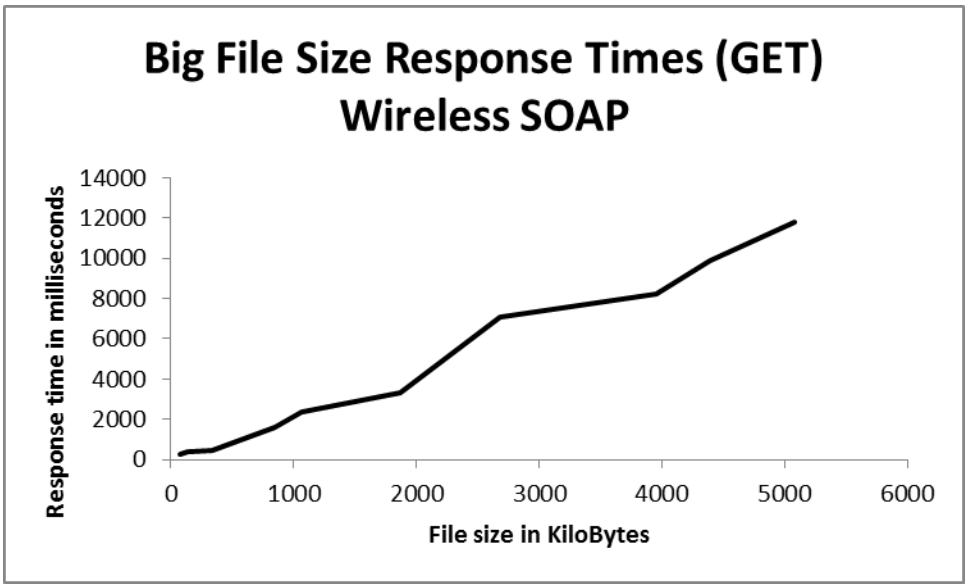

Figure 25: Big File Size Response Times (GET) Wireless SOAP

6.1.4.1.3 Response Time by File Size Comparison REST vs. SOAP

Figures 26 and 27 show the comparison for REST vs. SOAP for response times by differing file sizes in a wired and then a wireless environment.

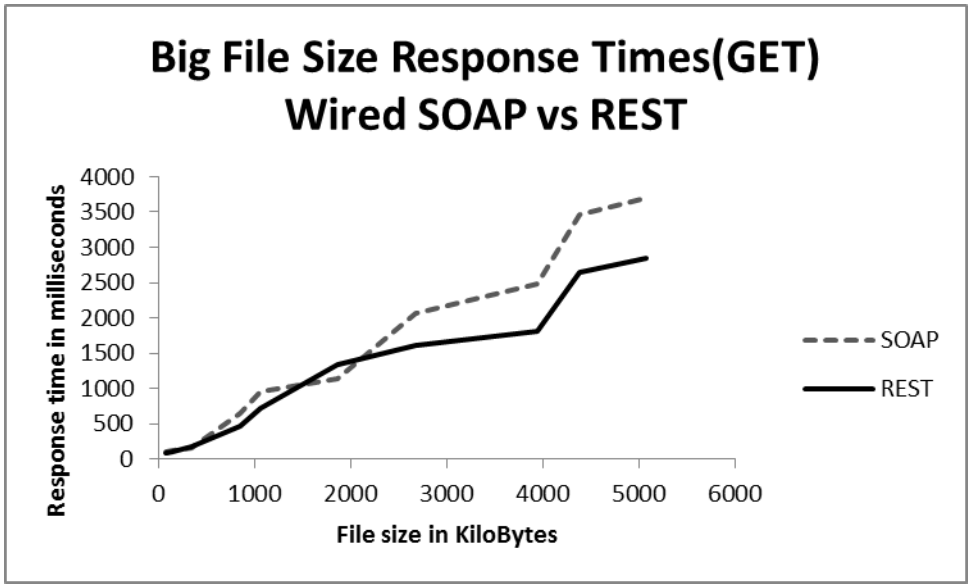

Figure 26: Big File Size Response Times (GET) Wired SOAP vs. REST

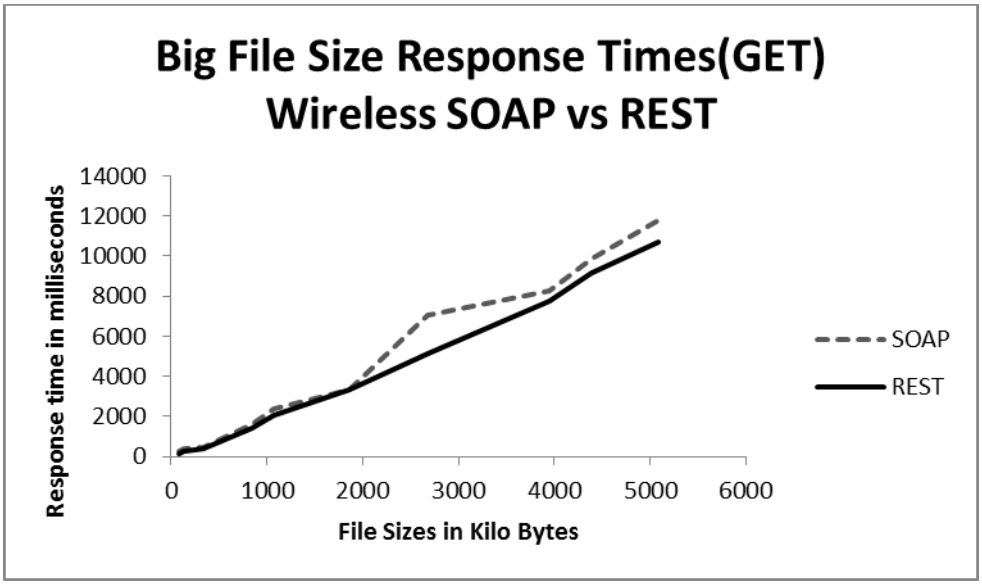

Figure 27: Big File Size Response Times (GET) Wireless SOAP vs. REST

REST response times were comparatively better than SOAP response times, which is in keeping with the general GET function response times with multiple clients discussed before.

6.1.4.2 Throughput in KiloBytes per Second

Throughput in KiloBytes was calculated by dividing file size in KiloBytes by the response time in seconds. Figures 30 through 31 represent file size in KiloBytes on the x-axis and the throughput in KiloBytes per second on the y-axis.

6.1.4.2.1 Wired Network

Figures 28 and 29 plot the throughput of REST and SOAP when dealing with differing file sizes in a wired environment.

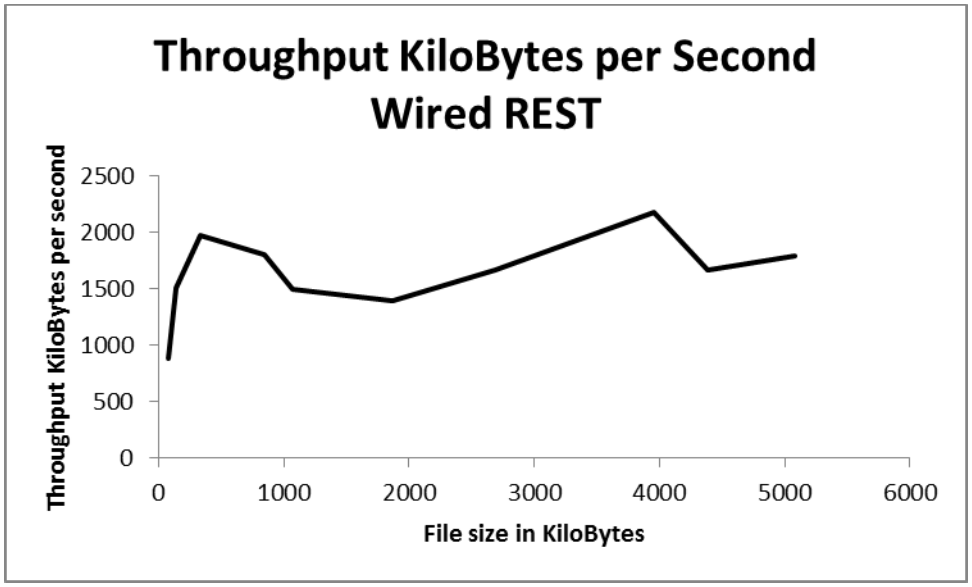

Figure 28: Throughput KiloBytes per Second Wired REST

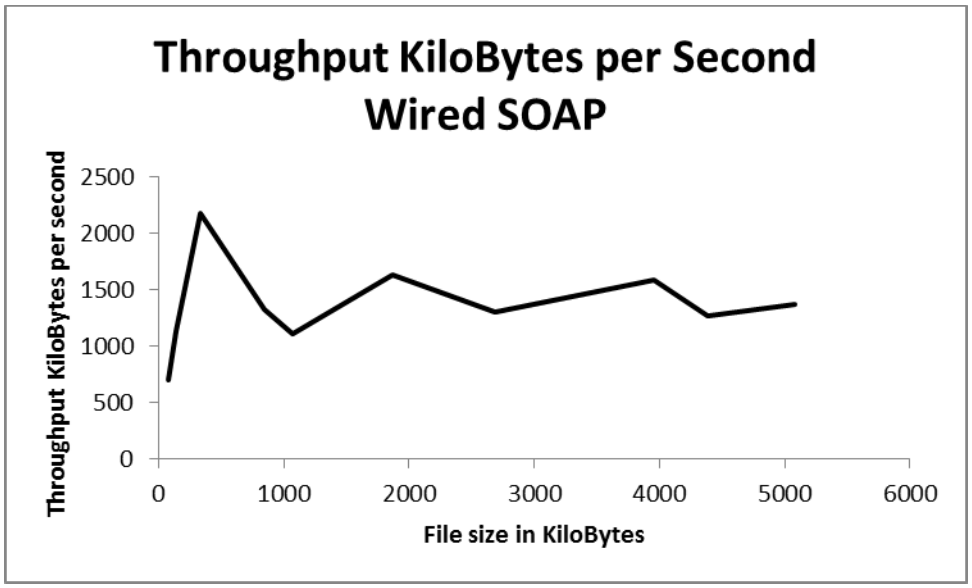

Figure 29: Throughput KiloBytes per Second Wired SOAP

6.1.4.2.2 Wireless Network

Figures 30 and 31 plot the throughput of REST and SOAP when dealing with differing file sizes in a wireless environment.

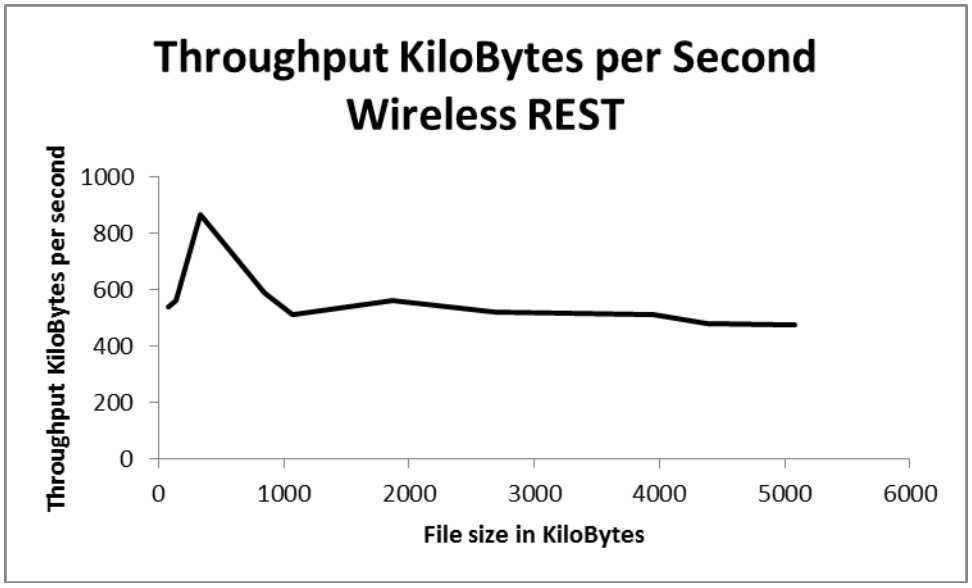

Figure 30: Throughput KiloBytes per Second Wireless REST

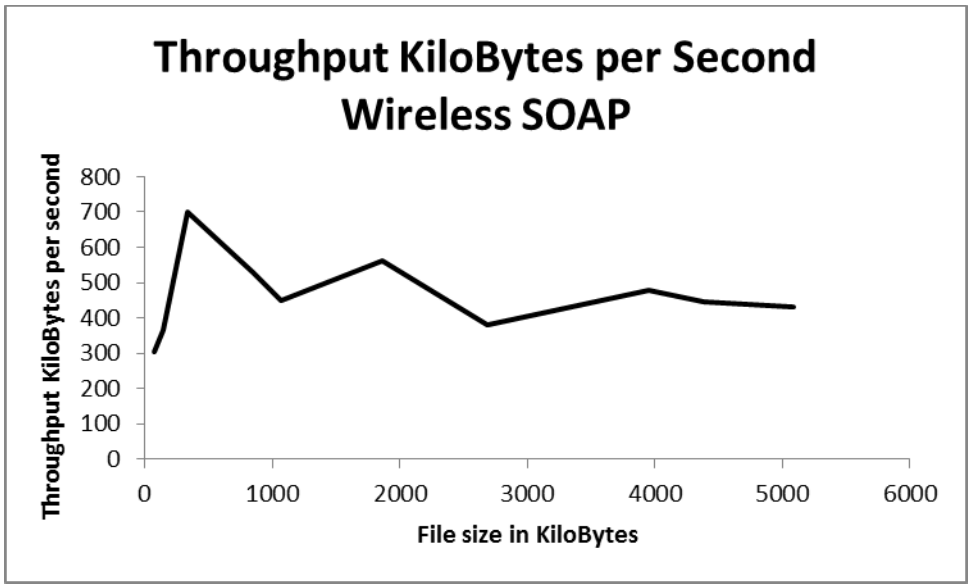

Figure 31: Throughput KiloBytes per Second Wireless SOAP

6.1.4.2.3 Throughput in KiloBytes per Second Comparison REST vs. SOAP

Figures 32 and 33 show the comparison for REST vs. SOAP of throughput in KiloBytes pers second in a wired and then a wireless environment.

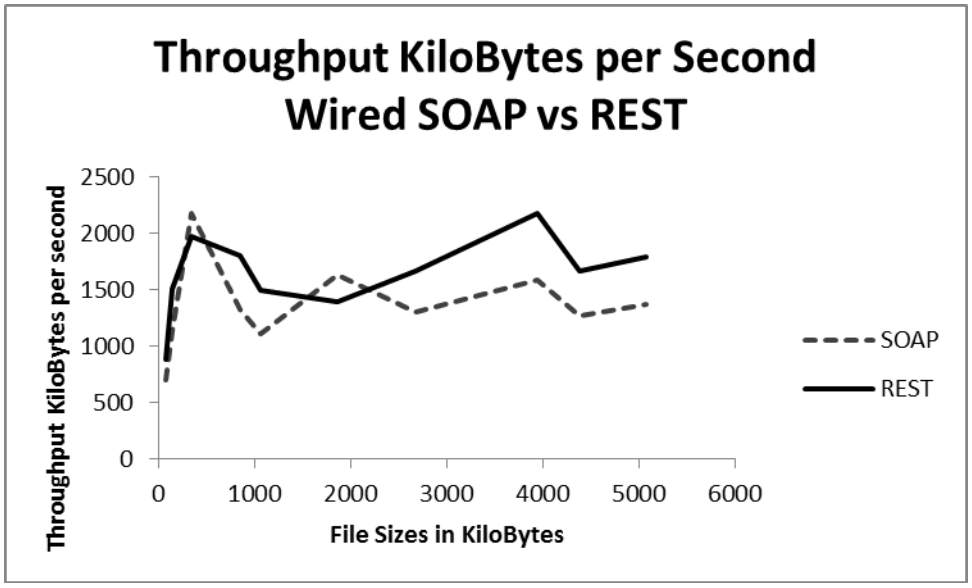

Figure 32: Throughput KiloBytes per Second Wired SOAP vs. REST

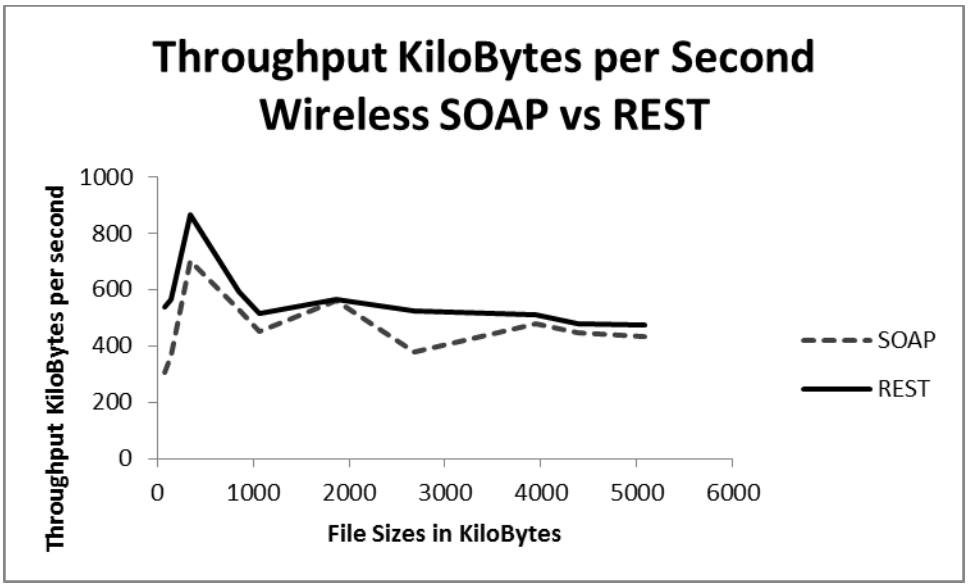

Figure 33: Throughput KiloBytes per Second Wireless SOAP vs. REST

Figues 32 and 33 illustrate that REST performance was comparatively better than that of SOAP. This is a request we might have expected, given earlier results.

6.1.4.3 Throughput in Clients per Second

Throughput, expressed as clients per second, was calculated by dividing the number of clients by the response time in seconds. The values for response times were taken from the results for the GET function with multiple clients, as discussed before. Figures 34 through 38 represent the number of simultaneous clients on the x-axis and throughput in clients per second on the y-axis.

6.1.4.3.1 Wired Network

Figures 34 and 35 plot the throughput of REST and SOAP when dealing with differing numbers of simultaneous service requests in a wired environment.

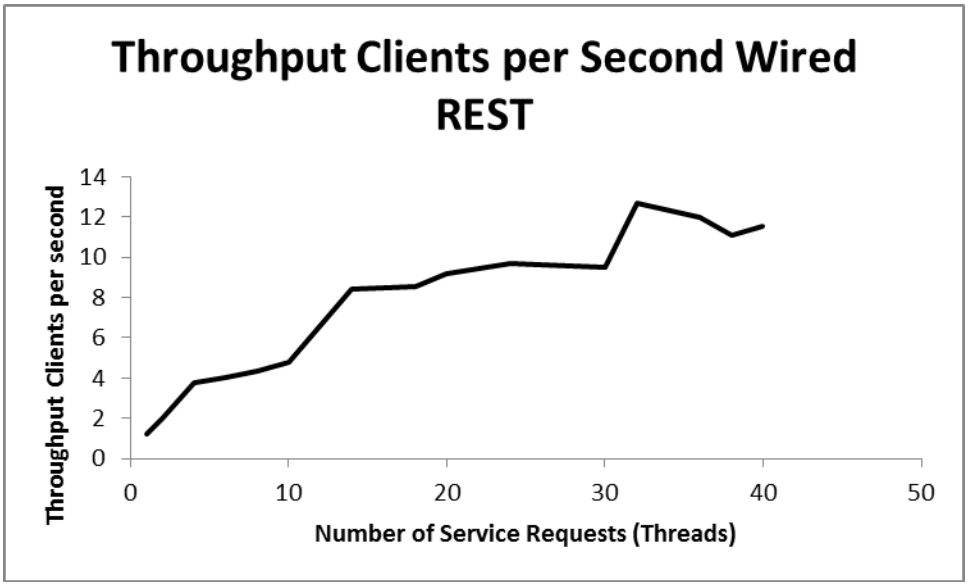

Figure 34: Throughput Clients per Second Wired REST

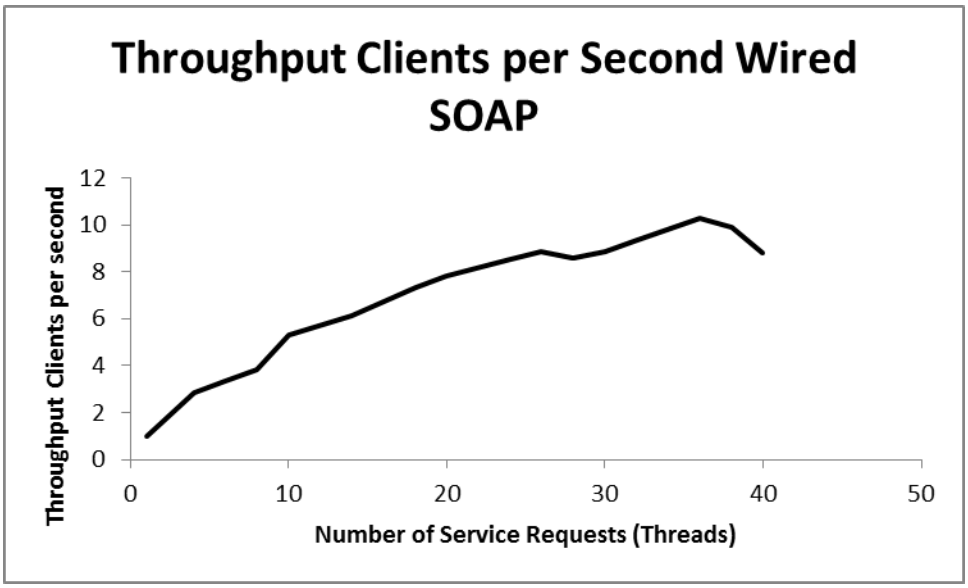

Figure 35: Throughput Clients per Second Wired SOAP

6.1.4.3.2 Wireless Network

Figures 36 and 37 plot the throughput of REST and SOAP when dealing with differing numbers of simultaneous service requests in a wireless environment.

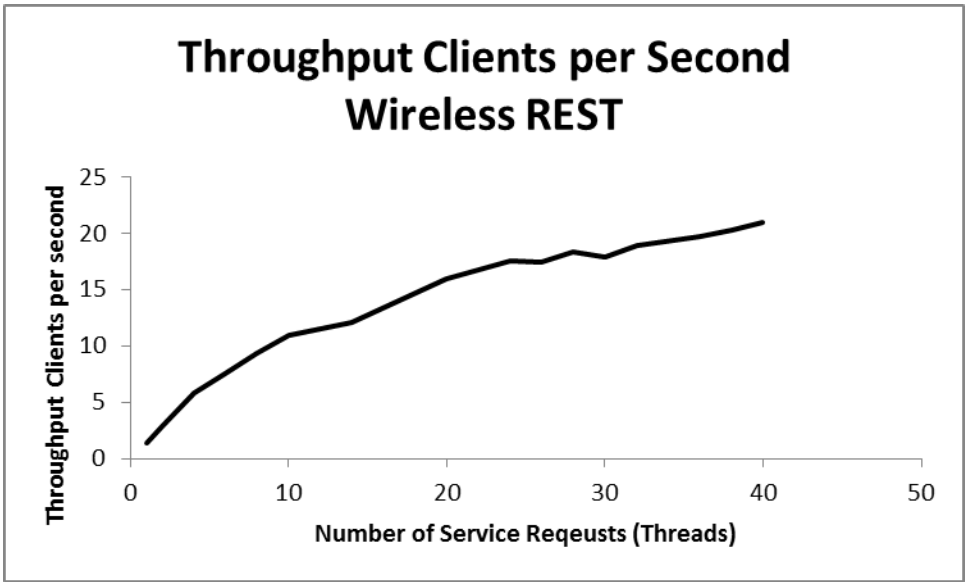

Figure 36: Throughput Clients per Second Wireless REST

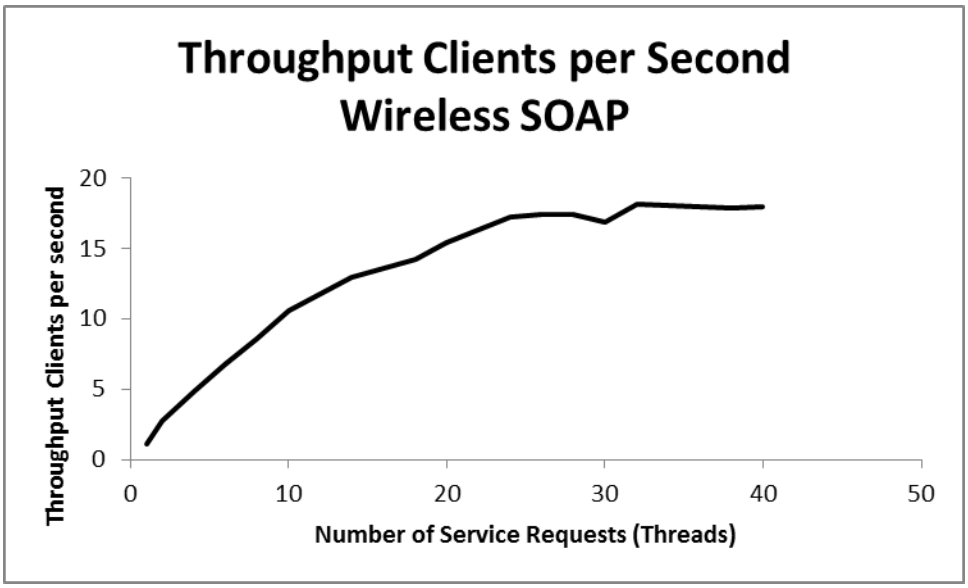

Figure 37: Throughput Clients per Second Wireless SOAP

6.1.4.3.3 Throughput in Clients per Second Comparison REST vs. SOAP

Figures 38 and 39 show the comparison of REST versus SOAP of throughput in clients per second in a wired and then a wireless environment.

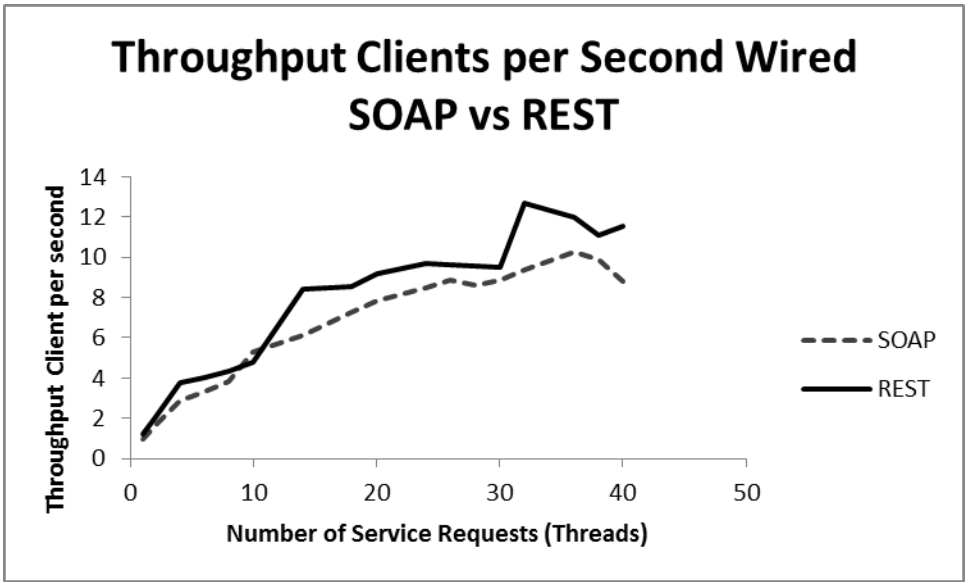

Figure 38: Throughput Clients per Second Wired SOAP vs. REST

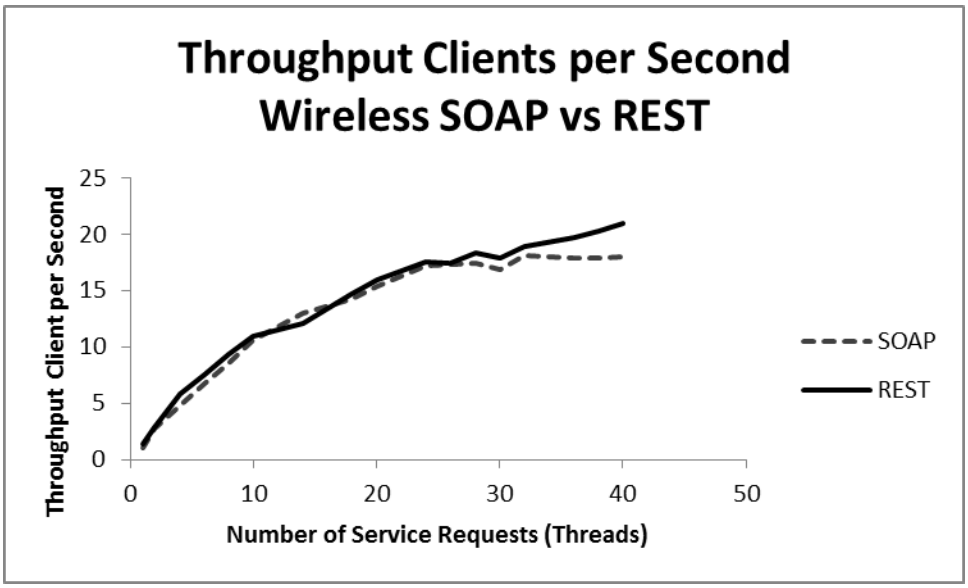

Figure 39: Throughput Clients per Second Wireless SOAP vs. REST

REST performance was comparatively better than that of SOAP from the above graphs and got better as the number of service requests increased.

6.2 Statistical Significance

Tests for statistical significance tell us whether there is the probability of a relationship between two variables or if they are merely some random variables. In other words, statistical significance means there is a good chance we are right in finding a relationship exists between two findings [Walonick97]. No findings can be 100 percent certain; they depend on various factors and observations. Often we are satisfied if our findings are more than 90 percent certain that the observed result is true. We select a generally acceptable level for our findings to have occurred by chance (e.g., 0.10/10% or 0.05/5%). This is called the alpha (i.e., chosen significance level or confidence level). The probability of observing a result by chance is usually called a P value. Results are said to be statistically significant, if the calculated P value is less than the alpha value, otherwise they are insignificant. Many management or behavioral studies accept the 0.05 level of significance, thus we chose our alpha value to be 0.05.

There are different types of statistical significance tests. T-tests are generally used to test whether there are differences between two groups on the same variable. The T-test assesses whether the means of two groups are statistically different from each other. This analysis is appropriate whenever you want to compare the means of two groups [Trochim06]. We chose to perform paired sample T-tests because our data collection systems were independent. Table 2 shows the results of the T- tests we conducted for

different experiments. With the alpha set to 0.05, the P values are all less than the alpha, showing our results were statistically significant.

	Paired Samples Test Paired Differences 95% Confidence Interval of the Difference				
Experiment (REST vs SOAP)	Lower	Upper	t	df	Sig (2- Tailed) P-value < 0.05, the result is statistically significant
GET Wireless	-144.97741	-44.96201	-4.026	16	.001
GET Wired	-531.14030	-234.03499	-5.460	16	.000
POST Wired	-975.67853	-743.29735	-15.681	16	.000
POST Wireless	-361.90263	-274.29784	-15.395	16	.000
All Four Functions Wired	-4116.14566	-2240.69554	-7.270	14	.000
All Four Functions Wireless	-568.66732	-270.36996	-6.033	14	.000
Throughput bytes per second Wired	-476.20537	-72.71803	-3.078	9	.013
Throughput bytes per second Wireless	-155.84645	-40.15715	-3.833	9	.004
Throughput Clients per second Wired	.66531	1.64842	4.989	16	.000
Throughput Clients per second Wireless	.34403	1.28167	3.676	16	.002

Table 2: Statistical Significance

Chapter 7

CONCLUSION

The evaluation of SOAP vs. REST was difficult because coding with REST was challenging. In comparing the two technologies, we found SOAP is well developed with much industrial support but REST is still striving to establish itself and its importance. This comment on the two technologies is apt:

> "REST is more an old philosophy than a new technology. Whereas SOAP looks to jump-start the next phase of Internet development with a host of new specifications, the REST philosophy espouses that the existing principles and protocols of the Web are enough to create robust Web services." [Asarawala02]

When the IDE and the tools are kept apart, REST is easy to study. Understanding REST is easier, if one has knowledge of HTTP, because of their similar architectures. REST requires immense amounts of coding effort at the server; at client levels, coding is comparatively easy. The author took nearly a year to learn coding with REST, utilizing all the functions it supports. The author went through different IDEs and application servers for designing the REST Web service. That many software engineers are not aware of REST is a known fact.

SOAP is often easy to code, because of the magnitude of support available and the fact that so many are familiar with the product. Unlike REST, SOAP server side coding is simple, but the client side is difficult.

In this study, the response times of REST were better than those of SOAP. Performance depends on many factors, such as the development and IDE support. It is interesting that REST proved to be considerably faster, even though its use is limited by simple IDE support and not much development support.

A study of mobile Web services also proved REST has better response time [Hamad10]. Yahoo, Google, EBay, and Amazon are running their Web services on REST. Amazon runs its Web services on both SOAP and REST and claims that the performance of REST is better than that of SOAP. REST consumes less bandwidth compared to SOAP and is a preferred choice for lightweight applications. For applications that require high bandwidth and complex processing SOAP is a better choice.

REST has experienced a lot of development and has been frequently studied since 2009, after the advent of Jersey and Sun JAX-RS Web services built over Jersey. Security is something that depends on the application developer; SOAP and REST are technologies specifically for building Web services. Currently, the only security measure that REST offers is HTTPS, which is not sufficient or near the security offered by SOAP. HTTPS is a good way of transferring data but it will cover only the transmission path. During this transmission, there is no control of the actions on the data until it reaches HTTPS and after it reaches the destination [Bertocci05]. Without any idea of how the data is processed before and after the HTTPS channel, REST does not offer better options for security; this is where SOAP stack encryption appears to have the advantage.

Recent versions of SOAP and WSDL (Web Services Description Language) have been designed to enable more RESTful use of Web services, since it has been recognized that REST is essential to accommodating the growing needs of Web applications [Chintaka07]. Google and Amazon apply security to their REST interfaces using developer tokens, which is a reinvented functionality that is present in SOAP/WS-Security [Web2.0security10]. This, combined with the fact that many changes in SOAP enable RESTful use and that much development is going on in REST (Low REST, High REST) to allow support of complex Web applications, indicates SOAP is increasingly utilizing REST principles in its design and REST is working on supporting complex web services. Currently, a Web service design depends on the needs of a particular business. Moreover, Web services now are being divided into RESTful Web services and Big Web services.

The results of this study and the statements by Sudhanshu [Sudhanshu09] allow us to conclude that REST seems to have more scalability, interoperability, and performance compared to SOAP, whereas SOAP has more security and reliability.

REFERENCES

Print Publications:

[Aldea] Costel Aldea, Livia Sangeorzan, and Alina Aldea, "Web Services and Enterprise Games," Proceedings of the 9th WSEAS International Conference on Simulation, Modeling, and Optimization.

[Cherkasova03] Ludmila Cherakasova, Yun Fu, Wenting Tang, and Amin Vahdat, "Measuring and Characterizing End-to-End Internet Service Performance," ACM Transactions on Internet Technology, Vol. 3, No. 4, November 2003, pp. 347-391.

[Fielding00] Dr. Roy Thomas Fielding - doctoral thesis, "Architectural Styles and the Design of Network-Based Software Architecture," Department of Computer and Information Science, University of California, Irvine 2000.

[Hamad10] Hatem Hamad, Motaz Saad, and Ramzi Abed, "Performance Evaluation of RESTful Web Services for Mobile Devices," International Arab Journal of e-Technology, Vol.1, No. 3, January 2010.

[Liu04] Yutu Liu, Anne H. H. Ngu, and Liangzhao Zeng, "QoS Computation and Policing in Dynamic Web Service Selection," WWW 2004, May 17-22, New York, USA.

[Muehlen05] Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson, "Developing Web Services Choreography Standards - The Case of REST vs. SOAP," 2004, Elsevier B.V.

[Newcomer02] Eric Newcomer, "Understanding Web Services: XML, WSDL, SOAP, and UDDI," Addison-Wesley Professional (May 23, 2002).

[Richardson07] Leonard Richardson, Sam Ruby, and David Heinemeie, "Restful Web Services – Leonard Richardson," Sam Ruby O'Reilly Publications (May 8, 2007).

Electronic Sources:

[Answers11] "Web Services Document," http://www.answers.com/topic/web-service, last accessed October 13, 2010.

[Asarawala02] Amit Asarawala, "Giving SOAP a REST," http://www.devx.com/DevX/Article/8155, last accessed March 4, 2011.

[Bertocci05] Vittorio Bertocci, "End to End Security," http://blogs.msdn.com/b/vbertocci/archive/2005/04/25/end-to-end-security-or-why-you-shouldn-t-drive-your-motorcycle-naked.aspx?CommentPOSTed=true#commentmessage, last accessed March 13, 2011.

[Booth04] David Booth, Hugo Haas, Francis McCabe, et al., "Web Services Architecture," http://www.w3.org/TR/ws-arch/, last accessed March 20, 2011 .

[Chintaka07] Eran Chinthaka, "Enable REST with Web Services, Part 1: REST and Web Services in WSDL 2.0," http://www.ibm.com/developerworks/webservices/library/ws-rest1/, last accessed March 7, 2011.

[Christensen 01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, "WSDL 1.1 Document," www.w3.org/TR/wsdl, last accessed March 7, 2011.

[REST11] "Representational State Transfer Document," http://en.wikipedia.org/wiki/Representational_State_Transfer, last accessed February 20, 2011.

[Rodriguez08] Alex Rodriquez, "RESTful Web Services: The Basics," https://www.ibm.com/developerworks/webservices/library/ws-restful/, last accessed October 10, 2010.

[SOAP11] "SOAP Document," http://en.wikipedia.org/wiki/SOAP, last accessed March 2, 2011.

[Sudhanshu09] Sudhanshu, "How I explained REST to a SOAP pro," http://www.infosysblogs.com/microsoft/2009/08/how_i_explained_rest_to_a_soap.html, last accessed March 5, 2011.

[Trochim06] William M.K. Trochim, "The T-Test," http://www.socialresearchmethods.net/kb/stat_t.php, last accessed on February 7, 2011.

[Walonick97] David S. Walonick, "A Selection from Survival Statistics," http://www.statpac.com/surveys/surveys.pdf, last accessed February 10, 2011.

[Web2.0security10] "Security for REST and Web2.0," www.matsays.com/downloads/inf400/inf400-sp10-wk14b2.pdf, last accessed March 10, 2011.

Appendix A

Server Code (REST and SOAP)

REST Application Server:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package restresources;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.Consumes;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import java.sql.*;
import restresources.Customers;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.URI;
import java.util.Properties;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.StreamingOutput;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import com.sun.jersey.api.NotFoundException;
```

```
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.PathParam;
import javax.ws.rs.Consumes;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;

/**
 * REST Web Service
 *
 * @author n00446144
 */

@Path("/restserver")
public class RestServer {
    @Context
    private UriInfo context;

    /** Creates a new instance of RestServer */
    public RestServer() {
    }


    @GET
    @Produces("text/html")
    public String getHtml() throws Exception {


            DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
                    System.out.println("Drivers loaded: ");
                    Connection
con=DriverManager.getConnection("jdbc:odbc:kumar","SYSTEM","pavan");
                    System.out.println("Connection Established:");
                    //PreparedStatement ps=con.prepareStatement("select * from custs where custid=101");
                    PreparedStatement ps=con.prepareStatement("select * from custs1 ");
                    ResultSet rs=ps.executeQuery();
                    rs.next();
 ps.close();
con.close();
rs.close();

// have to check this...

        return"<html><body><h1><p> " +"First name: " +rs.getString(1)+" <br> "+"Last
Name:"+"<br>"+rs.getString(2)+" <br> "+"id:"+rs.getString(3)+"<br> "+"Salary:"+rs.getString(4)+ "<br>
" +"City:"+rs.getString(5)+"<br> " +"Country:"+rs.getString(6)+ " <br>" +"Mobile:"+rs.getString(7)+  "
<br>" +"Status"+rs.getString(8)+ "<br> " +"Email:"+rs.getString(9)+ " <br>" +"Ssn:"+rs.getString(10)+"
<br>" +"ZipCode"+rs.getString(11)+ "<br>"+"</p></body></h1></html>";



    }
```

```
  // from now second GET....
    @GET
   @Path ("{id}")
   @Produces ("text/html")
   public  String getHTML(@PathParam ("id") String customerId)throws Exception {

      DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
                        System.out.println("Drivers loaded11: ");
                        Connection
con=DriverManager.getConnection("jdbc:odbc:kumar","SYSTEM","pavan");
                        System.out.println("Connection Established11:");
                        //PreparedStatement ps=con.prepareStatement("select * from custs where custid=101");
                        PreparedStatement ps=con.prepareStatement("select * from custs1 where
customerid="+customerId);
                        ResultSet rs=ps.executeQuery();
                        rs.next();
           //            return"<html><body><h1><p> "+rs.getString(1)+"  "+rs.getString(2)+"
"+rs.getString(3)+" "+rs.getString(4)+"</p></body></h1></html>";

             return"<html><body><h1><p> "+rs.getString(1)+"  "+rs.getString(2)+"  "+rs.getString(3)+"
"+rs.getString(4)+ " " +rs.getString(5)+" " +rs.getString(6)+ " " +rs.getString(7)+  " " +rs.getString(8)+ " "
+rs.getString(9)+ " " +rs.getString(10)+" " +rs.getString(11)+ "</p></body></h1></html>";



      }



   @POST
   @Consumes ("application/xml")

   //public Response addCustomer(InputStream customerData) throws ParserConfigurationException,
SAXException, IOException {

   //(changes removed the Response keyword and then commented the return and added void to the method
   public  String addCustomer(InputStream customerData) throws ParserConfigurationException,
SAXException, IOException {

      try {
         int cid=01;
         String fname="one",lname="two",zcode="three";
         int sal=02,mob=03;
         String cit="two",coun="three",stat="four",mail="five",sn="six";


         DocumentBuilder documentBuilder =
         DocumentBuilderFactory.newInstance().newDocumentBuilder();
         Document document = documentBuilder.parse(customerData);
         document.getDocumentElement().normalize();

         NodeList nodeList = document.getElementsByTagName("customer");

         Node customerRoot = nodeList.item(0);
```

```
if (customerRoot.getNodeType() == Node.ELEMENT_NODE) {
   Element element = (Element) customerRoot;
   NodeList childNodes = element.getChildNodes();
   for (int i = 0; i < childNodes.getLength(); i++) {
            Element childElement = (Element)childNodes.item(i);
      String tagName = childElement.getTagName();
      String textContent = childElement.getTextContent();
      if(tagName.equals("id")){
         cid=Integer.parseInt(textContent);
      }
      else if (tagName.equals("firstname")) {
         fname=textContent;
         //customer.setFirstName(textContent);
      } else if (tagName.equals("lastname")) {
         lname=textContent;
         //customer.setLastName(textContent);
      } else if (tagName.equals("zipcode")) {
         zcode=textContent;
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("salary")) {
         sal=Integer.parseInt(textContent);
         //customer.setZipcode(textContent);
      }
    else if (tagName.equals("mobile")) {
         mob=Integer.parseInt(textContent);
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("city")) {
         cit=textContent;
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("country")) {
         coun=textContent;
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("status")) {
         stat=textContent;
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("email")) {
         mail=textContent;
         //customer.setZipcode(textContent);
      }
 else if (tagName.equals("ssn")) {
         sn=textContent;
         //customer.setZipcode(textContent);
      }
   }
}
   //Customers customer = buildCustomer(null, customerData);
       //long customerId = persist(customer, 0);
       //return Response.created(URI.create("/" + customerId)).build();
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con=DriverManager.getConnection("jdbc:odbc:kumar","system","pavan");
```

```
                    PreparedStatement ps=con.prepareStatement("Insert into custs1
values(?,?,?,?,?,?,?,?,?,?,?)");
            ps.setString(1,fname);
                    ps.setString(2,lname);
                    ps.setInt(3,cid);
                    ps.setInt(4,sal);
            ps.setString(5,cit);
            ps.setString(6,coun);
            ps.setInt(7,mob);
            ps.setString(8,stat);
            ps.setString(9,mail);
            ps.setString(10,sn);
            ps.setString(11,zcode);
                    ps.executeUpdate();
ps.close();
con.close();

      return"operation successful";
       }


       catch (Exception e) {
                throw new WebApplicationException(e, Response.Status.INTERNAL_SERVER_ERROR);
              }


   }




   @PUT
  @Path("{id1}")
   @Consumes ("application/xml")

   // taken post and made changes reading id from customer using the path param id-String customerId then
updating using prepared statements

   public String updateCustomer(@PathParam("id1") String customerId1, InputStream input) throws
ParserConfigurationException, SAXException, IOException {

       try {
        //   int cid=01;
          String fname="one",lname="two",zcode="three";

          int sal=02,mob=03;
          String cit="two",coun="three",stat="four",mail="five",sn="six";
          int cid=0;

          DocumentBuilder documentBuilder =
          DocumentBuilderFactory.newInstance().newDocumentBuilder();
          Document document = documentBuilder.parse(input);
          document.getDocumentElement().normalize();

          NodeList nodeList = document.getElementsByTagName("customer");
```

```
Node customerRoot = nodeList.item(0);
if (customerRoot.getNodeType() == Node.ELEMENT_NODE) {
   Element element = (Element) customerRoot;
   NodeList childNodes = element.getChildNodes();
   for (int i = 0; i < childNodes.getLength(); i++) {
            Element childElement = (Element)childNodes.item(i);
      String tagName = childElement.getTagName();
      String textContent = childElement.getTextContent();

      if(tagName.equals("id")){
         cid=Integer.parseInt(textContent);
      }
      else if (tagName.equals("firstname")) {
         fname=textContent;
         //customer.setFirstName(textContent);
      } else if (tagName.equals("lastname")) {
         lname=textContent;
         //customer.setLastName(textContent);
      } else if (tagName.equals("zipcode")) {
         zcode=textContent;
         //customer.setZipcode(textContent);
      }

    else if (tagName.equals("salary")) {
         sal=Integer.parseInt(textContent);
         //customer.setZipcode(textContent);
      }
     else if (tagName.equals("mobile")) {
         mob=Integer.parseInt(textContent);
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("city")) {
         cit=textContent;
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("country")) {
         coun=textContent;
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("status")) {
         stat=textContent;
         //customer.setZipcode(textContent);
      }
   else if (tagName.equals("email")) {
         mail=textContent;
         //customer.setZipcode(textContent);
      }
 else if (tagName.equals("ssn")) {
         sn=textContent;
         //customer.setZipcode(textContent);
      }
```

```
            }

          }

            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    Connection con=DriverManager.getConnection("jdbc:odbc:kumar","system","pavan");


                    PreparedStatement ps1=con.prepareStatement("Update custs1 set
firstname=?,lastname=?,customerid=?,salary=?,city=?,country=?,mobile=?,status=?,email=?,ssn=?,zipcode
=? where customerid="+customerId1);

                    ps1.setString(1, fname);
          ps1.setString(2, lname);
          ps1.setInt(3, cid);
          ps1.setInt(4, sal);
          ps1.setString(5, cit);
          ps1.setString(6, coun);
          ps1.setInt(7, mob);
          ps1.setString(8, stat);
          ps1.setString(9, mail);
          ps1.setString(10, sn);
          ps1.setString(11,zcode);
                    ps1.executeUpdate();
ps1.close();
con.close();
       return "Operation Successful";
       }


       catch (Exception e) {
                 throw new WebApplicationException(e, Response.Status.INTERNAL_SERVER_ERROR);
             }
           }
// here adding the code for delete

  @DELETE
    @Path("{id}")
    public String deleteCustomer(@PathParam("id") String customerId) {
                    try {

                              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    System.out.println("drivers loaded:");
                    Connection con=DriverManager.getConnection("jdbc:odbc:kumar","system","pavan");
                    System.out.println("Connection Established:");
                    //int cid1=30;
                    PreparedStatement ps1=con.prepareStatement("delete from custs1 where customerid =
?");
                    int cid1=Integer.parseInt(customerId);
                    ps1.setInt(1,cid1);
                    System.out.println("coming to here:");
                    int i=ps1.executeUpdate();
                    System.out.println(i+"  rows deleted");
                    ps1.close();
             con.close();
```

```
                return "operation Successful";
            } catch (Exception e) {
                        throw new WebApplicationException(e,
Response.Status.INTERNAL_SERVER_ERROR);
                }
    }
}
```

Customers.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package restresources;

/**
 *
 * @author aryan
 */
public class Customers {
    private Integer customerId;
        private String firstName;
        private String lastName;
        private String zipcode;
      private String salary;
      private String city;
      private String country;
      private Integer mobile;
      private String status;
      private String email;
      private String ssn;
      //private String zipcode;



        public Integer getCustomerId() {
                return customerId;
        }
        public void setCustomerId(Integer customerId) {
                this.customerId = customerId;
        }
        public String getFirstName() {
                return firstName;
        }
        public void setFirstName(String firstName) {
                this.firstName = firstName;
        }
        public String getLastName() {
                return lastName;
        }
        public void setLastName(String lastName) {
                this.lastName = lastName;
        }
        public String getZipcode() {
```

```
		return zipcode;
	}
	public void setZipcode(String zipcode) {
		this.zipcode = zipcode;
	}

    public String getSalary() {
		return salary;
	}
	public void setCustomerId(String salary) {
		this.salary = salary;
	}
 public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
 public String getCountry() {
		return country;
	}
	public void setCountry(String country) {
		this.country = country;
	}

    public Integer getMobile() {
		return mobile;
	}
	public void setMobile(Integer mobile) {
		this.mobile = mobile;
	}
public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}

    public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
public String getSsn() {
		return ssn;
	}
	public void setSsn(String ssn) {
		this.ssn = ssn;
	}

}
```

Exceptions.java

```
import java.util.MissingResourceException;
import java.util.ResourceBundle;

public class Messages {
	private static final String BUNDLE_NAME = "messages"; //$NON-NLS-1$

	private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle
			.getBundle(BUNDLE_NAME);

	private Messages() {
	}

	public static String getString(String key) {
		try {
			return RESOURCE_BUNDLE.getString(key);
		} catch (MissingResourceException e) {
			return '!' + key + '!';
		}
	}
}
```

JDBC Connection.java

```
package restresources;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class JDBCConnection {
	private static PreparedStatement ps;
	private static Connection con;

	public static Connection getConnection() throws SQLException, ClassNotFoundException{
		Class.forName(Messages.getString("JDBCConnection.0"));

	con=DriverManager.getConnection(Messages.getString("JDBCConnection.1"),Messages.getStrin
g("JDBCConnection.2"),Messages.getString("JDBCConnection.3"));

			return con;
	}

	public static PreparedStatement getPreparedStatement(String sqlStatement) throws
SQLException, ClassNotFoundException{
			Connection con=getConnection();
			ps=con.prepareStatement(sqlStatement);

			return ps;
	}

	public static void closePreparedStatement() throws SQLException{
			ps.close();
	}

	public static void closeConnection() throws SQLException{
			ps.close();
```

```
                con.close();
        }
}
```

For Throughputs:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package restresources;
import java.io.BufferedInputStream;
import javax.imageio.*;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.StreamingOutput;
import javax.ws.rs.PUT;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.Consumes;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import java.sql.*;
//import HelloWorldResource1.Customers;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.URI;
import java.util.Properties;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.StreamingOutput;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
```

```
import com.sun.jersey.api.NotFoundException;
import java.awt.Image;
import java.awt.image.BufferedImage;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.PathParam;
import javax.ws.rs.Consumes;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 * REST Web Service
 *
 * @author aryan
 */

@Path("/restserver/image42")
public class imagetest4 {
    @Context
    private UriInfo context;

    /** Creates a new instance of imagetest1 */
    public imagetest4() {
    }


  @GET
 @Path("{id}")
//@Produces ("StreamingOutput/text")

public byte[] GettheFile(@PathParam ("id") String cId) throws ParserConfigurationException,
SAXException, IOException, Exception, java.io.IOException,
                WebApplicationException


    {

    try {

        InputStream in = null;
            File file = new File("C:/"+cId+".png");

            in = new BufferedInputStream(new FileInputStream(file));



        FileInputStream fis = new FileInputStream(file);
```

```
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        byte[] buf = new byte[1024];
        try {
           for (int readNum; (readNum = fis.read(buf)) != -1;) {
              bos.write(buf, 0, readNum);

              System.out.println("read " + readNum + " bytes,");
           }
        } catch (IOException ex) {
           //Logger.getLogger(ConvertImage.class.getName()).log(Level.SEVERE, null, ex);

        System.out.println("exception caught :"+ex);
        }
byte[] bytes = bos.toByteArray();



return bytes;
    }
    catch (Exception e) {
                throw new WebApplicationException(e, Response.Status.INTERNAL_SERVER_ERROR);
             }

    }
}
```

SOAP Application Server:

SOAPApplicationServer

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.soapserver;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 *
 * @author aryan
 */
@WebService()
public class SoapServer {

    @WebMethod(operationName = "addCustomers")
    public String addCustomer(@WebParam(name = "customer")    Customer customer) throws
WebApplicationException
 {
        try {
            //TODO write your implementation code here:
        // Pre Initializing the variables
            int cid = 01;
            String fname = "one";
            String lname = "two";
            String zcode = "three";
            int sal = 02;
            int mob = 03;
            String cit = "two";
            String coun = "three";
            String stat = "four";
            String mail = "five";
            String sn = "six";
           // Logger.getAnonymousLogger().info(customer.getId());
            Logger.getAnonymousLogger().info(customer.getSsn());
            cid = customer.getId();
            fname = customer.getFirstname();
            lname = customer.getLastname();
            zcode = customer.getZipcode();
```

```
            sal = customer.getSalary();
            mob =customer.getMobile();
            cit = customer.getCity();
            coun = customer.getCountry();
            stat = customer.getStatus();
            mail = customer.getEmail();
            sn = customer.getSsn();
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection("jdbc:odbc:kumar", "system", "pavan");
            PreparedStatement ps = con.prepareStatement("Insert into custs1 values(?,?,?,?,?,?,?,?,?,?,?)");
            ps.setString(1, fname);
            ps.setString(2, lname);
            ps.setInt(3, cid);
            ps.setInt(4, sal);
            ps.setString(5, cit);
            ps.setString(6, coun);
            ps.setInt(7, mob);
            ps.setString(8, stat);
            ps.setString(9, mail);
            ps.setString(10, sn);
            ps.setString(11, zcode);
            ps.executeUpdate();
            ps.close();
            con.close();
            return "Test Success";
        } catch (SQLException ex) {
            Logger.getLogger(SoapServer.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(SoapServer.class.getName()).log(Level.SEVERE, null, ex);
        }

//        catch(java.io.IOException e)
//{
//     throw new WebApplicationException(e.getMessage());
// }
  return "Test Success";
 }



    /**
     * Web service operation
     */
    @WebMethod(operationName = "updateCustomers")
    public String updateCustomers(@WebParam(name = "customer")
    Customer customer, @WebParam(name = "id")int id) throws WebApplicationException{
        try {
            //TODO write your implementation code here:
            //parsing the value of customer id to be updated from a integet value to a local variable
            int custid = id;
            // Pre Initializing the variables
            int cid = 01;
            String fname = "one";
            String lname = "two";
            String zcode = "three";
            int sal = 02;
```

```
            int mob = 03;
            String cit = "two";
            String coun = "three";
            String stat = "four";
            String mail = "five";
            String sn = "six";
            // Logger.getAnonymousLogger().info(customer.getId());
           // Logger.getAnonymousLogger().info(customer.getSsn());
            cid = customer.getId();
            fname = customer.getFirstname();
            lname = customer.getLastname();
            zcode = customer.getZipcode();
            sal = customer.getSalary();
            mob = customer.getMobile();
            cit = customer.getCity();
            coun = customer.getCountry();
            stat = customer.getStatus();
            mail = customer.getEmail();
            sn = customer.getSsn();
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection("jdbc:odbc:kumar", "system", "pavan");
            PreparedStatement ps1 = con.prepareStatement("Update custs1 set
firstname=?,lastname=?,customerid=?,salary=?,city=?,country=?,mobile=?,status=?,email=?,ssn=?,zipcode
=? where customerid=" + id);
            System.out.println("the value for customerid to be updated:"+id);
            ps1.setString(1, fname);
            ps1.setString(2, lname);
            ps1.setInt(3, cid);
            ps1.setInt(4, sal);
            ps1.setString(5, cit);
            ps1.setString(6, coun);
            ps1.setInt(7, mob);
            ps1.setString(8, stat);
            ps1.setString(9, mail);
            ps1.setString(10, sn);
            ps1.setString(11,zcode);
            int executeUpdate = ps1.executeUpdate();
            System.out.println("the value of eupdate: "+executeUpdate);

            ps1.close();
            con.close();
            return "Test Success";
            //return null;
        } catch (SQLException ex) {
            Logger.getLogger(SoapServer.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(SoapServer.class.getName()).log(Level.SEVERE, null, ex);
        }
    return "Test Success";

    }

@WebMethod(operationName = "deleteCustomers")
    public String deleteCustomers(@WebParam(name = "id1")int id1) throws WebApplicationException
{
        try {
```

```
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("drivers loaded:");
            Connection con = DriverManager.getConnection("jdbc:odbc:kumar", "system", "pavan");
            System.out.println("Connection Established:");
            //int cid1=30;
            PreparedStatement ps2 = con.prepareStatement("delete from custs1 where id = ?");
            ps2.setInt(1, id1);
            System.out.println("coming to here:");
            int i = ps2.executeUpdate();
            System.out.println(i + "  rows deleted");

            ps2.close();
            con.close();
            //return "Test Success";
        } catch (SQLException ex) {
            Logger.getLogger(SoapServer.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(SoapServer.class.getName()).log(Level.SEVERE, null, ex);
        }
return "Test Sucess and the Customer data was deleted";
}
}
```

Exception.java

```
package org.soapserver;

public class WebApplicationException extends Exception {
            public WebApplicationException() {
                        // TODO Auto-generated constructor stub

                        super();
            }

            public WebApplicationException(String message) {
                        // TODO Auto-generated constructor stub

                        super(message);
            }
}
```

Customer.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.soapserver;

/**
 *
 * @author aryan
 */
public class Customer {
            private int id;
            private String firstname;
            private String lastname;
```

```
private String zipcode;
private int salary;
private String city;
private String country;
private int mobile;
private String status;
private String email;
private String ssn;
public int getId() {
	return id;
}
public void setId(int id) {
	this.id = id;
}
public String getFirstname() {
	return firstname;
}
public void setFirstname(String firstname) {
	this.firstname = firstname;
}
public String getLastname() {
	return lastname;
}
public void setLastname(String lastname) {
	this.lastname = lastname;
}
public String getZipcode() {
	return zipcode;
}
public void setZipcode(String zipcode) {
	this.zipcode = zipcode;
}
public int getSalary() {
	return salary;
}
public void setSalary(int salary) {
	this.salary = salary;
}
public String getCity() {
	return city;
}
public void setCity(String city) {
	this.city = city;
}
public String getCountry() {
	return country;
}
public void setCountry(String country) {
	this.country = country;
}
public int getMobile() {
	return mobile;
}
public void setMobile(int mobile) {
	this.mobile = mobile;
}
```

```
	public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getSsn() {
		return ssn;
	}
	public void setSsn(String ssn) {
		this.ssn = ssn;
	}

}
```

SOAP Code for Throughputs:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package att.test;

import javax.jws.WebService;
import java.io.File;
import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 *
 * @author n00446144
 */
@WebService()
public class NewWebService {
@WebMethod(operationName = "atttest")
   public DataHandler atttest( @WebParam(name = "id")    int id) throws WebApplicationException

{
    File file = new File("C:/"+id+".png");

   FileDataSource theImageDataSource = new FileDataSource(file);
DataHandler theImageDataHandler = new DataHandler(theImageDataSource);
```

```
return theImageDataHandler;


}
}
```

WebException.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package att.test;

/**
 *
 * @author Ultramatics
 */
public class WebApplicationException extends Exception {
        public WebApplicationException() {
                // TODO Auto-generated constructor stub

                super();
        }

        public WebApplicationException(String message) {
                // TODO Auto-generated constructor stub

                super(message);
        }
}
```

Appendix B

Client Code (REST and SOAP)

REST Client (Same for both Wired and Wireless Clients):

Main Class:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package restapplicationserverclientwireless;
import restapplicationserverclientwireless.RestClientImpl;


/**
 *
 * @author n00446144
 */
public class Main {

    private static int numberOfThreads=20;
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Thread[] thread=new Thread[numberOfThreads];

        for(int i=0;i<numberOfThreads;i++){
            thread[i]=new Thread(new RestClientImpl(),"thread"+i);
            thread[i].start();
        }
    }

}
```

RestClientImpl.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package restapplicationserverclientwireless;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;
```

```
import java.io.ByteArrayInputStream;

import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.DeleteMethod;
import org.apache.commons.httpclient.methods.InputStreamRequestEntity;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.PutMethod;
import org.apache.commons.httpclient.methods.RequestEntity;
/**
 *
 * @author n00446144
 */
public class RestClientImpl implements Runnable {

 private static int correlationId = 800;

    private static synchronized int getCorrelationId() {
        return correlationId++;
    }

    public void run() {
        try {
          //  Logger.getAnonymousLogger().info("Number of threads="+Thread.activeCount());
            int correlationId = RestClientImpl.getCorrelationId();
          // testGetCustomer(correlationId);
            testAddCustomer(correlationId);
//           //changing the following function without passing correlationid
            testGetCustomer(correlationId);
            testUpdateCustomer(correlationId);
            testDeleteCustomer(correlationId);

        } catch (Exception ex) {
            Logger.getLogger(RestClientImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void testAddCustomer(int correlationId) throws Exception {

        Logger.getAnonymousLogger().info("Entry Add Customer@"+(new Date()).getTime()+" for
correlationId=" + correlationId+" Thread Name="+Thread.currentThread().getName());
        //Logger.getAnonymousLogger().info("Number of threads="+Thread.activeCount());

        final String addCustomerXML =
                 "<customer>" +
                    "<id>/correlationId</id>"+
                "<firstname>5Joe121</firstname>" +
                    "<lastname>5Schmo121</lastname>" +
                    "<zipcode>59804821</zipcode>" +
                    "<salary>598776</salary>"+
                "<city>5hyd</city>"+
                "<country>5india</country>"+
                "<mobile>59887656</mobile>"+
                "<status>5Active</status>"+
                "<email>5pk@yahoo.com</email>"+
                "<ssn>598876-98</ssn>"+
               "</customer>";
```

```
            PostMethod postMethod = new PostMethod("http://139.62.166.47:8080/RestApplication-
Server/resources/restserver");
            RequestEntity entity = new InputStreamRequestEntity(new
ByteArrayInputStream(addCustomerXML.getBytes()), "application/xml");
           postMethod.setRequestEntity(entity);
            HttpClient client = new HttpClient();
            try {
               int result = client.executeMethod(postMethod);

            } finally {
               postMethod.releaseConnection();
            }



      //System.out.println("the value in test1 :"+test1);
      //Logger.getAnonymousLogger().info("Exit AddCust@"+(new Date()).getTime()+" for
correlationId=" + correlationId+" Thread Name="+Thread.currentThread().getName());
   }

   public void testGetCustomer(int correlationId) throws Exception {
      // Logger.getAnonymousLogger().info("Entry GET@"+(new Date()).getTime()+" for correlationId="
+ correlationId+" Thread Name="+Thread.currentThread().getName());
      // Logger.getAnonymousLogger().info("Number of threads="+Thread.activeCount());
       final String BASE_URI = "http://139.62.166.47:8080/RestApplication-Server/resources/";
//          System.out.println("the value of string"+BASE_URI);
//           System.out.println("coming to here before start of client");
   Client c = Client.create();
   WebResource service = c.resource(BASE_URI);


  String in;
 //changing the following line adding 612 later change to 6120
  in=service.path("/restserver/gettest/"+correlationId).get(String.class);
 //in=service.path("/restserver/gettest/612").get(String.class);
System.out.println("the value of String 'in' : "+in);
      //Logger.getAnonymousLogger().info("Exit GET @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
   }


   public void testUpdateCustomer(int correlationId) throws Exception {
     //  Logger.getAnonymousLogger().info("Entry for correlationId=" + correlationId+" Thread
Name="+Thread.currentThread().getName());
// Logger.getAnonymousLogger().info("Entry Update@"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
       final String addCustomerXML =
                 "<customer>" +
                    "<id>/correlationId</id>"+
                "<firstname>11Joe121</firstname>" +
                    "<lastname>5Schmo121</lastname>" +
                    "<zipcode>59804821</zipcode>" +
                    "<salary>598776</salary>"+
                "<city>5hyd</city>"+
                "<country>5india</country>"+
                "<mobile>59887656</mobile>"+
```

```
                "<status>5Active</status>"+
                "<email>5pk@yahoo.com</email>"+
                "<ssn>598876-98</ssn>"+
              "</customer>";

            PutMethod putMethod = new PutMethod("http://139.62.166.47:8080/RestApplication-
Server/resources/restserver/6130");
            RequestEntity entity = new InputStreamRequestEntity(new
ByteArrayInputStream(addCustomerXML.getBytes()), "application/xml");
            putMethod.setRequestEntity(entity);
            HttpClient client = new HttpClient();
            try {
               int result = client.executeMethod(putMethod);
               System.out.println("Response status code: " + result);

              /*  System.out.println("Response headers:");
               Header[] headers = putMethod.getResponseHeaders();
               for (int i = 0; i < headers.length; i++) {
                  System.out.println(headers[i].toString());
               }*/
            } finally {
               putMethod.releaseConnection();
            }
      // Logger.getAnonymousLogger().info("Exit for correlationId=" + correlationId+" Thread
Name="+Thread.currentThread().getName());

    //Logger.getAnonymousLogger().info("Exit Update @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
   }

   public void testDeleteCustomer(int correlationId) throws Exception {

      // Logger.getAnonymousLogger().info("Entry for correlationId=" + correlationId+" Thread
Name="+Thread.currentThread().getName());
       // Logger.getAnonymousLogger().info("Entry Delete@"+(new Date()).getTime()+" for
correlationId=" + correlationId+" Thread Name="+Thread.currentThread().getName());
      DeleteMethod deleteMethod = new DeleteMethod("http://139.62.166.47:8080/RestApplication-
Server/resources/restserver/"+correlationId);
            HttpClient client = new HttpClient();
            try {
               int result = client.executeMethod(deleteMethod);
               System.out.println("Response status code: " + result);
            } finally {
       deleteMethod.releaseConnection();
            }
      // Logger.getAnonymousLogger().info("Exit for correlationId=" + correlationId+" Thread
Name="+Thread.currentThread().getName());
    Logger.getAnonymousLogger().info("Exit Delete @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
   }
}
```

REST Client for Calculating Throughput :

```
package imagenewone;
```

```
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
//import com.sun.jersey.multipart.BodyPart;
//import com.sun.jersey.multipart.MultiPart;
//import jersey.multipart.demo.model.Project;
import java.awt.image.BufferedImage;
import org.apache.commons.httpclient.HttpException;
import java.sql.*;
import java.io.*;
import javax.imageio.*;
import sun.net.www.content.image.png;
import javax.ws.rs.WebApplicationException;
/**
 *
 * @author n00446144
 */
public class Main
{
   public static void main(String s[]) throws
java.sql.SQLException,WebApplicationException,java.lang.ClassNotFoundException
          {
  try {
   Main.testgettingfiles();
//   RESTClient.testUpdateCustomer();
//   RESTClient.testDeleteCustomer();
  } catch (HttpException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }

 }
 private static void testgettingfiles() throws IOException, WebApplicationException, HttpException,
java.sql.SQLException,java.lang.ClassNotFoundException,
com.sun.jersey.api.client.UniformInterfaceException {
//System.out.println("coming to here, start of the getmethod in client");
// this is the original one          final String BASE_URI = "http://localhost:8080/RestApplication-
Server/resources";
final String BASE_URI = "http://139.62.166.204:8080/RestApplication-Server/resources";
          //final String BASE_URI = "http://139.62.166.47:8080/RestApplication-Server/resources";
  //        System.out.println("the value of string"+BASE_URI);
    //        System.out.println("coming to here before start of client");
    Client c = Client.create();
    WebResource service = c.resource(BASE_URI);
    //System.out.println("coming to here after baseuri");
    //System.out.println("the time before fetching starts:");
long g= System.currentTimeMillis();
// this is the original oneInputStream in=service.path("/restserver/image12").get(InputStream.class);
System.out.println("Starting Time :"+g);
byte[] in=service.path("/restserver/image42/5").get(byte[].class);
//InputStream in=service.path("/restserver/image12").get(InputStream.class);
long f=System.currentTimeMillis();
System.out.println("End time:"+f);
```

```
System.out.println("total processing time :"+(f-g));
//OutputStream out = new FileOutputStream("c:/6666687.png");
//       out.write(in);

       }
}
```

SOAP Clients:
Main Class :

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package soapserverclient;

/**
 *
 * @author n00446144
 */
public class Main {

    private static int numberOfThreads=20;
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Thread[] thread=new Thread[numberOfThreads];

        for(int i=0;i<numberOfThreads;i++){
            thread[i]=new Thread(new SoapClientImpl(),"thread"+i);
            thread[i].start();
        }
    }

}
```

SOAPClientImpl.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package soapserverclient;

import com.att.Customer;
import com.att.NewWebService;
import com.att.NewWebServiceService;
import com.att.WebApplicationException_Exception;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author n00446144
 */
public class SoapClientImpl implements Runnable {

    private static int correlationId = 800;

    private static synchronized int getCorrelationId() {
        return correlationId++;
    }
```

```
    public void run() {
        try {
            Logger.getAnonymousLogger().info("Number of threads="+Thread.activeCount());
            int correlationId = SoapClientImpl.getCorrelationId();
          // testGetCustomer(correlationId);
             testAddCustomer(correlationId);
           //changing the following function without passing correlationid
            testGetCustomer(correlationId);
            testUpdateCustomer(correlationId);
            testDeleteCustomer(correlationId);

        } catch (WebApplicationException_Exception ex) {
            Logger.getLogger(SoapClientImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void testAddCustomer(int correlationId) throws WebApplicationException_Exception {

        Logger.getAnonymousLogger().info("Entry @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
        //Logger.getAnonymousLogger().info("Number of threads="+Thread.activeCount());
        NewWebServiceService newService = new NewWebServiceService();
        NewWebService newWS = newService.getNewWebServicePort();

        String test1 = "null";

        Customer mycustomer = new Customer();
        mycustomer.setCity("Jacksonville");
        mycustomer.setCountry("USA");
        mycustomer.setEmail("kumar@jax.com");
        mycustomer.setFirstname("pavan");
        mycustomer.setId(correlationId);
        mycustomer.setLastname("potti");
        mycustomer.setMobile(87769989);
        mycustomer.setSalary(8000);
        mycustomer.setSsn("878-654");
        mycustomer.setStatus("active");
        mycustomer.setZipcode("32256");

        test1 = newWS.addCustomers(mycustomer);
       //System.out.println("the value in test1 :"+test1);
       // Logger.getAnonymousLogger().info("Exit AddCust@"+(new Date()).getTime()+" for
correlationId=" + correlationId+" Thread Name="+Thread.currentThread().getName());
    }

    public void testGetCustomer(int correlationId) throws WebApplicationException_Exception {
        //Logger.getAnonymousLogger().info("Entry GET@"+(new Date()).getTime()+" for correlationId="
+ correlationId+" Thread Name="+Thread.currentThread().getName());
       // Logger.getAnonymousLogger().info("Number of threads="+Thread.activeCount());
        NewWebServiceService newService = new NewWebServiceService();
        NewWebService newWS = newService.getNewWebServicePort();
        String res = null;
        //String ten="10";
        //following is the original line..changing this for get testing..will be reverted for general 4 functions
testing
```

```
        res = newWS.getCustomers(correlationId);

        // the following is the new line
              // res = newWS.getCustomers(800);
        System.out.println("the result came :" + res);
       // Logger.getAnonymousLogger().info("Exit GET @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
    }

//     public void testGetCustomer(int correlationId) throws WebApplicationException_Exception {
//        long g;
//
//         Logger.getAnonymousLogger().info("Entry @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
//        //Logger.getAnonymousLogger().info("Number of threads="+Thread.activeCount());
//        NewWebServiceService newService = new NewWebServiceService();
//        NewWebService newWS = newService.getNewWebServicePort();
//        String res = null;
//        //String ten="10";
//        //following is the original line..changing this for get testing..will be reverted for general 4 functions
testing
//        //res = newWS.getCustomers(correlationId);
//
//        // the following is the new line
//              res = newWS.getCustomers(800);
//        System.out.println("the result came :" + res);
//        Logger.getAnonymousLogger().info("Exit @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
//    }
    public void testUpdateCustomer(int correlationId) throws WebApplicationException_Exception {
      //  Logger.getAnonymousLogger().info("Entry for correlationId=" + correlationId+" Thread
Name="+Thread.currentThread().getName());
// Logger.getAnonymousLogger().info("Entry Update@"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
        NewWebServiceService newService = new NewWebServiceService();
        NewWebService newWS = newService.getNewWebServicePort();
        Customer mycustomer1 = new Customer();
        mycustomer1.setCity("Jacksonville");
        mycustomer1.setCountry("USA");
        mycustomer1.setEmail("kumar@jax.com");
        mycustomer1.setFirstname("pavan");
        mycustomer1.setId(correlationId);//changing here testing
        mycustomer1.setLastname("potti");
        mycustomer1.setMobile(87769989);
        mycustomer1.setSalary(8000);
        mycustomer1.setSsn("878-654");
        mycustomer1.setStatus("active");
        mycustomer1.setZipcode("32256");
        String res1 = null;
        res1 = newWS.updateCustomers(mycustomer1, correlationId);

        System.out.println(res1);
       // Logger.getAnonymousLogger().info("Exit for correlationId=" + correlationId+" Thread
Name="+Thread.currentThread().getName());
```

```
   // Logger.getAnonymousLogger().info("Exit Update @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
   }

   public void testDeleteCustomer(int correlationId) throws WebApplicationException_Exception {

      // Logger.getAnonymousLogger().info("Entry for correlationId=" + correlationId+" Thread
Name="+Thread.currentThread().getName());
       // Logger.getAnonymousLogger().info("Entry Delete@"+(new Date()).getTime()+" for
correlationId=" + correlationId+" Thread Name="+Thread.currentThread().getName());
       NewWebServiceService newService = new NewWebServiceService();
       NewWebService newWS = newService.getNewWebServicePort();
       String ar2 = null;
       ar2 = newWS.deleteCustomers(correlationId);
       System.out.println(ar2);
      // Logger.getAnonymousLogger().info("Exit for correlationId=" + correlationId+" Thread
Name="+Thread.currentThread().getName());

    Logger.getAnonymousLogger().info("Exit Delete @"+(new Date()).getTime()+" for correlationId=" +
correlationId+" Thread Name="+Thread.currentThread().getName());
   }
}
SOAPServerService.java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package soapserverclient;

/**
 *
 * @author n00446144
 */
class SoapServerService {

   public SoapServerService() {
   }

}



SOAP code for Throughput Measurement:
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package soapimageserverclient;

import att.test.NewWebService;
Import att.test.NewWebServiceService;
import att.test.WebApplicationException_Exception;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
```

```
import java.io.IOException;
import java.io.OutputStream;

/**
 *
 * @author n00446144
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws WebApplicationException_Exception,
FileNotFoundException, IOException {
        // TODO code application logic here


    NewWebServiceService newService = new NewWebServiceService();
            NewWebService newWS = newService.getNewWebServicePort();

            byte[] buf=null;
            long g=System.currentTimeMillis();
            System.out.println("Start Time :"+g);

            buf=newWS.atttest(10);
“
            long f=System.currentTimeMillis();
            System.out.println("End Time :"+f);

            System.out.println(" The Total Process Time :"+(f-g));

OutputStream out=new FileOutputStream("C:/10.png");
out.write(buf);

   out.flush();
    }

}
```

VITA

Pavan Kumar Potti earned a bachelor of technology degree in electronics and communications from Jawaharlal Nehru Technological University, Hyderabad, India, in 2004 and expects to receive his master of science in computer science from the University of North Florida in spring 2011. Pavan Kumar worked as an embedded systems engineer for two years at D'gipro Systems, in Bangalore, India, prior to starting graduate work. Pavan Kumar worked as a graduate assistant and research assistant at the University of North Florida. He also worked as a Google Ambassador for the University of North Florida, from spring 2008 to fall 2009. Pavan Kumar also served as a president for the UNF IEEE-CS and AITP student chapters, from spring 2008 to fall 2009. He received a Best Graduate Research Student award from the University of North Florida, for spring 2009. Pavan Kumar has a journal publication on RFID and a couple of international conference publications on wireless security and wireless network design.

Pavan Kumar worked as a software engineer at EverBank for 13 months and is concentrating now on his studies on network security. His ongoing interests include Web services and REST programming, and he has started developing applications for Google Android 3.0. Pavan Kumar has a very good knowledge of embedded systems and programming experience in C, C++, Java Web Services, and Oracle. He enjoys playing cricket, Ping-Pong, and chess, and lately has developed a passion for Xbox 360. Pavan Kumar is single, never married.