University of Toronto

Comparative Study of Web Service Architectures for Software Development Kit of Transportation Applications

by

Ta Jiun Ting

A Thesis submitted to the
Faculty of Applied Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Applied Science

Department of Civil Engineering

Toronto, ON
April 2019

Abstract

The use of Web services is an effective way to transfer information over the Web on demand, suitable for transportation applications where requests are made from different locations. However, there is a cost associated with each request in terms of the computational resources used. Therefore, this paper seeks to recommend a Web service architecture by comparing REST and SOAP, the two prevalent approaches to building Web services.

This paper explores the two different styles by comparing their performance when used in the transportation research context. First, this paper establishes the metrics that are compared and reviews similar studies in other areas of research. Afterward, tests are conducted by developing RESTful and SOAP-based Web services that perform identical functions in a cloud computing environment. Finally, the performance is measured using Azure API Management, the API management solution created by Microsoft.

As part of this thesis, 5 Web services were selected in accordance with the needs of the research team at the University of Toronto Transportation Research Institute, including functions such as data processing and retrieval. These Web services and the database are hosted on Microsoft Azure, and the associated APIs are published using Azure API Management. Subsequently, tests are designed to measure the performance of these APIs in terms of speed and resource usage. Lastly, the analysis is performed using the monitoring tools provided by Azure API Management to obtain the results and recommendations for future implementations in this area.

The analysis results show that RESTful Web services use less bandwidth than their SOAP-based counterparts. However, there is no significant difference between the two architectures in terms of response time. Therefore, a RESTful approach to Web services is recommended, as the SOAP-based design has seemingly no advantage over the RESTful approach.

Acknowledgements

I would like to express my most sincere gratitude to my thesis supervisor Dr. Baher Abdulhai of the Faculty of Applied Science and Engineering at the University of Toronto. He has provided me with continuous support throughout my undergraduate study and the door to his office is always open when I needed guidance.

I would also like to thank Ph.D. Candidate Ahmed Aqra for his unrelenting support throughout the final two years of my undergraduate study, as well as the writing of this paper. Without his extensive knowledge and passionate input, the research on Web service performance could not have been successfully conducted.

Finally, I must also offer my deepest gratitude to my parents for their unwavering support and constant encouragement throughout my undergraduate study and writing this thesis. I must also thank Xun Wei for all her support. This accomplishment would not have been possible without them.

Author

Ta Jiun Ting

Table of Contents

List of Tables

List of Figures

Nomenclature

%	Percent
API	Application programming interface
ATIS	Advanced Traveler Information System
BPEL	Business Process Execution Language
CPU	Central processing unit
CRUD	Create, read, update, and delete
CSV	Comma-separated values
ESB	Enterprise service bus
GB	Gigabyte
GPS	Global Positioning System
GTA	Greater Toronto Area
HTTP	Hypertext Transfer Protocol
ITS	Intelligent Transportation System
ITSoS	Intelligent Transportation System of Systems
JSON	JavaScript Object Notation
ms	Millisecond
OSM	OpenStreetMap
REST	Representational state transfer
SDK	Software Development Kit
SQL	Structured Query Language
URI	Uniform Resource Identifiers
UTTRI	University of Toronto Transportation Research Institute

WSDL	Web Service Description Language
XML	Extensive Markup Language

Chapter 1 Introduction

Software development kit (SDK) is an essential tool for any software development environment, it enables the developers to contribute to the development process. Recently, researchers in the Intelligent Transportation System (ITS) research group at the University of Toronto Transportation Research Institute (UTTRI) started building an SDK for the Advanced Traveler Information System (ATIS). The ATIS project is part of the larger Intelligent Transportation System of Systems (ITSoS) project, and this study is part of the ATIS SDK.

1.1 Research Motivation

The goal of ATIS is to provide information to travellers to help them reach their destinations. Therefore, different types of information from multiple sources are needed to provide this service. One way to achieve this is to host all the data in a centralized database, then expose the required data using Web services. In this case, there would be a gateway that responds to specific requests made through an application programming interface (API), then serve the required data over Hypertext Transfer Protocol (HTTP).

1.2 Problem Definition

In order to meet the demands of a large system such as the ATIS, it is essential for the API to work correctly and efficiently. In addition, the research group has finite resources available across projects, thus the resource usage of the Web services should be minimized. Currently, there are two prevalent architectures used in Web services: SOAP (originally an acronym for Simple Object Access Protocol) and representational state transfer (REST). This paper explores the differences between the two approaches and compares the performance of these designs when deployed in the context of a transportation research group. Related work has been performed in other fields such as mobile devices and communications, but not in the transportation field. The results from related studies are presented in Chapter 2.

1.3 Research Methodology

To complete this comparison, five services are selected and outlined with based on the needs of the research group, and to represent the four basic operations used in database applications: create, read, update, and delete (CRUD). To ensure a fair comparison, Web services that provide identical functionalities are developed using both SOAP-based and RESTful approaches. The Web services are developed using Flask with Python 2.7, and the server is hosted on a virtual machine on Microsoft Azure. Azure API Management, the API management solution created by Microsoft, is used to record the performance of the Web services for this thesis. Lastly, performance is analyzed by repeatedly invoking the five services. More details regarding the Web services selected and the testing procedure are presented in Chapter 4.

1.4 Scope of Work

The purpose of this thesis is to investigate the two Web service architectures and recommend a design for the ATIS SDK. Therefore, performance is measured based on the five Web services recommended by the ATIS research group. In addition, three metrics are used during this comparison: request packet size, response packet size, and response time. These metrics are chosen based on a review of literature, and Azure API Management measures them with its built-in monitoring tool [1]. No additional tool is developed during this study to assess the performance of Web services. As such, the findings of this study are based solely on the comparison between the five Web services mentioned above using these three metrics. Chapter 4 provides more information regarding Azure API Management, as well as discusses the performance metrics in detail.

1.5 Assumptions

In order to simplify the problem at hand, this study relies on some assumptions to ensure a fair comparison between Web service architectures.

1. Python is used to develop Web services for both architectures. This paper assumes Python as a programming language does not favour any Web service architecture.
2. A third-party API management solution is used to measure the performance metrics for both architectures. This paper assumes the selected API management solution does not favour any architecture.
3. The Web services are developed using external open-source libraries and these libraries are treated as a black box. Different libraries are used to develop Web services for different architectures and may have different effects on performance between the two architectures. This paper assumes that the differences will be small enough to not affect the overall results.
4. The Web services will be hosted on a server, and a separate client will be used to test the API performance for both architectures. This paper assumes the hardware used during the comparison, such as the server and the client, does not favour any architecture.
5. All of the data that may be served by the API is stored in a back-end database for both architectures. This paper assumes the back-end database that the API is connected to does not favour any architecture.

1.6 Organization of Thesis

Chapter 2 reviews literature to establish what metrics should be measured and provides some results based on similar studies done in other fields. 00 provides more background information about API and API management, as well as provides a summary of the similarities and differences between the SOAP and RESTful design. Chapter 4 provides a detailed summary for each Web services and an explanation of the testing procedure. Chapter 5 examines the numerical results of the comparison, while Chapter 6 discusses the qualitative findings of the result. Finally, Chapter 7 outlines the recommendations or conclusions that can be drawn from the results of this comparison.

Chapter 2 Literature Review

This section reviews other studies that compare performance between SOAP-based and RESTful Web services in the past 10 years. While similar studies have been performed in other applications such as mobile devices and multimedia conferencing, there has been no such study in transportation-related applications. These studies provide a good foundation for evaluating the performance of SOAP-based and RESTful Web services, but it should be noted that results could differ depending on the application and its implementation.

Several recent studies [2], [3], [4] suggest that SOAP-based Web services have slower response times and larger network loads. Mulligan and Gracanin [2] conducted a study in 2009 to measure the performance of REST-based and SOAP-based middleware framework implementation using the Portal framework. The study found that REST-based implementations for middleware framework have better latency and smaller packet sizes than their SOAP-based counterparts [2]. The study also demonstrates that the difference in latency between the two implementations increases as the application becomes more complex, or as the number of synchronous requests increases [2]. A case study done in 2012 by Belqasmi et al. [3] reached similar conclusions when comparing the performance of Web services in multimedia conferencing. The case study compares the end-to-end delay of the conferencing operations, as well as the network load of the request and the response for each application [3]. The study found that SOAP-based Web services can take up to four times longer, with network loads up to three times compared to their RESTful counterparts in multimedia conferencing applications [3]. Lastly, a study conducted in 2012 by Mohamed and Wijesekera [4] that compares the performance of Web services on mobile devices also supports that RESTful Web services perform better than their SOAP-based counterparts in both response time and resource usage. The study compares Web services during concurrent requests, as well as the CPU and memory usage during testing [4]. The study concludes that SOAP-based implementations have a response time that is between 1.5 and 3 time longer, along with 2 to 5 times increased CPU consumption compared to RESTful Web services [4]. However, the finding is based on a single Web service tested using mobile devices, and the study does not draw any conclusions for other computing devices.

On the other hand, some studies [2], [5] show that support for each architecture varies depending on the implementation. In the enterprise environment, Kumari and Rath [5] claim that there is better support for SOAP-based implementations, and it is difficult to find the correct tool for REST-based implementations. In addition, the Enterprise service bus (ESB) tools used during the study did not "provide any means for integration of REST-based services" [5]. On the other hand, [2] notes that the SOAP-based implementation in the multimedia conferencing application requires the use of a specialized SOAP client to interact with a SOAP-enabled HTTP server. In addition, the SOAP framework used was "the only one that is still actively developed, open-source with a non-restrictive license" at the time of the study [2]. The study also mentions that REST-based implementation does not suffer from this restriction, and any HTTP client library to make requests to the REST server [2].

Chapter 3 Background

3.1 Application Programming Interface

An application programming interface (API) is a library of functions and tools for building other software. It is essential for any software development process, as it facilitates developers to break down larger problems into smaller tasks and subroutines that serve as building blocks to the overall project. In addition, an API can be used across multiple projects. This paper focuses on APIs that communicate over the Web, known as Web services. A Web service typically has exposed endpoints that respond to a set of predefined requests invoked over HTTP, then respond to the request by sending data using formats such as JavaScript Object Notation (JSON) or Extensive Markup Language (XML).

3.2 Architecture and Design

Currently, there are two main approaches to implementing Web services, SOAP-based and RESTful Web services. Each approach offers some advantages and disadvantages. Therefore, the choice of architecture may vary according to the application.

SOAP is a specification in which systems can interact with one another over the Web. The messages are sent in the XML format, usually over HTTP, but other transport protocol can be used as well [6]. It was developed in 1998 by Dave Winer et al. in Microsoft, to address the needs of large corporations and the enterprise market [7]. In SOAP-based implementations, Web services have a unique Web Service Description Language (WSDL) that is used to document the contract between the server and the client [5]. A SOAP message consists of a SOAP envelope, and the SOAP header/body are contained inside the envelope. SOAP envelope denotes the beginning and the end of the message, while the body contains the actual message being sent [8]. SOAP is a robust, extensive specification used by many systems across the world, and its use of open standards makes it easily extendable [4]. However, a SOAP message contains redundant information and “does not use many of the functionalities built into HTTP” [9].

On the other hand, representational state transfer (REST) is an architectural style created by Roy Fielding in the University of California, Irvine [10], and Web services conforming to the REST principles are known as RESTful services. RESTful services utilize GET, POST, PUT, and DELETE, the four basic built-in HTTP interaction methods, to directly access resources using Uniform Resource Identifiers (URI) [9]. Since RESTful services are built on top of mature Web standards widely in use, such as HTTP and URI, RESTful services are easier to provide services between organizations over the Web and are perceived to be more scalable [6]. In addition, RESTful services utilize the JSON format, which is lighter than XML format. Overall, RESTful services can run faster and utilize fewer resources, and they are freely accessible on the Web once deployed [4], [9]. However, it is difficult to transfer large and complex data as they need to be encoded into URI for them to be accessed [4], [9]. Table 3-1 contains a detailed comparison of the two design approaches discussed.

Table 3-1. Comparison between SOAP and REST, adapted from [7].

Topic	**SOAP**	**REST**
Pros	• Follows a formal enterprise approach • Works on top of any communication protocol, even asynchronously • Information about objects is communicated to clients • Security and authorization are part of the protocol • Can be fully described using WSDL	• Relatively easy to implement and maintain • Clearly separates client and server implementations • Communication is not controlled by a single entity • Information can be stored by the client to prevent multiple calls. • Can return data in multiple formats (JSON, XML etc.)

Table 3-2. (Continued)

Cons	• Costs a lot of bandwidth communicating metadata • Hard to implement	• Only works on top of the HTTP protocol • Hard to enforce authorization and security
When to use	• When clients need to have access to objects available on servers • When the goal is to enforce a formal contract between the client and the server	• When clients and servers operate on a Web environment • When information about objects does not need to be communicated to the client
When not to use	• When the goal is for most of the developers to easily use the API • When the bandwidth is very limited	• When there is a need to enforce a strict contract between client and server • When performing transactions that involve multiple calls
Conclusion	Use when dealing with transactional operations and the audience that is already satisfied with this technology	Use when the goal is wide-scale API adoption or when the API is targeted for mobile applications

3.3 API Lifecycle

A typical API has the following stages in its life cycle: development, analysis, operation, depreciated, and retired [7]. In the beginning, when the API is being designed and developed, it is not visible nor deployed [7]. Afterwards, the API is deployed to a limited set of consumers for them to try out and provide feedback, it is also analyzed for monetization in this stage [7]. Then, the API is fully in operation, where it is maintained, monitored, and scaled [7]. After some time, the API becomes depreciated when a new version is published, where it is still deployed but not

visible to new users [7]. Finally, the API becomes retired when it is unpublished and deleted [7]. This study investigates APIs that are in the development stage.

3.4 API Management

API management refers to the process of managing the API throughout its life cycle, from publishing APIs to ongoing maintenance. This is needed because the developer would like to assess the success of an API implementation according to a set of metrics to determine how the API is used. In addition, API management allows APIs to be more scalable, as it allows the developers to enforce usage and restrict access as the number of users increase. This is especially important since the API endpoint is exposed to the public. Moreover, API management provides documentation that allows developers of different background to learn about the API and start using it with ease [9]. With the recent surge of mobile and Web applications [11], more and more API management solutions are now available on the market. Azure API Management [12], the API management solution from Microsoft, is selected for this project.

3.5 API Performance

To assess the success of APIs, APIs need to be tested against a set of predefined metrics and criteria [9]. A load test is an effective strategy to determine the performance as well as resource utilization of APIs [9]. To do this, APIs should be put under test in load conditions, and the relevant metrics measured [9]. Common metrics include response time, latency, error rates, central processing unit (CPU) utilization, memory utilization, and message payload size [9]. On the other hand, security and user privacy are paramount for any Web service [13]. Aspects of API security testing include testing the authentication method, as well as testing against possible injection attacks [9].

Chapter 4 Methodology

As part of this thesis, five Web services are defined according to the needs of the research group. These Web services are invoked over HTTP to process requests or to update/retrieve pre-defined data. To create a comparison between architectures, two sets of Web services that perform identical functions are developed. The first set conforms to the RESTful style, while the second set is developed according to the SOAP standard. Afterwards, the performance comparison of these services is made using Microsoft Azure API Management Tool. The Web services are written in Python using the Flask Web framework [14], as well as other third-party tools and libraries to simplify the development process. As a result, a lot of the Web service interaction is abstracted away from the developers. In addition, the API server and the back-end database used are hosted on a cloud computing service, where the author has no direct control over how resources are utilized. Therefore, the results measured using these Web services may not reflect other RESTful and SOAP Web service implementations accurately.

4.1 Web Services in the ATIS SDK

This section discusses the APIs that are developed for this thesis, these APIs form part of the ATIS SDK. A preliminary design of the ATIS SDK was developed by Ahmed Aqra in June 2018 is shown in , with the portion related to this paper highlighted in red.

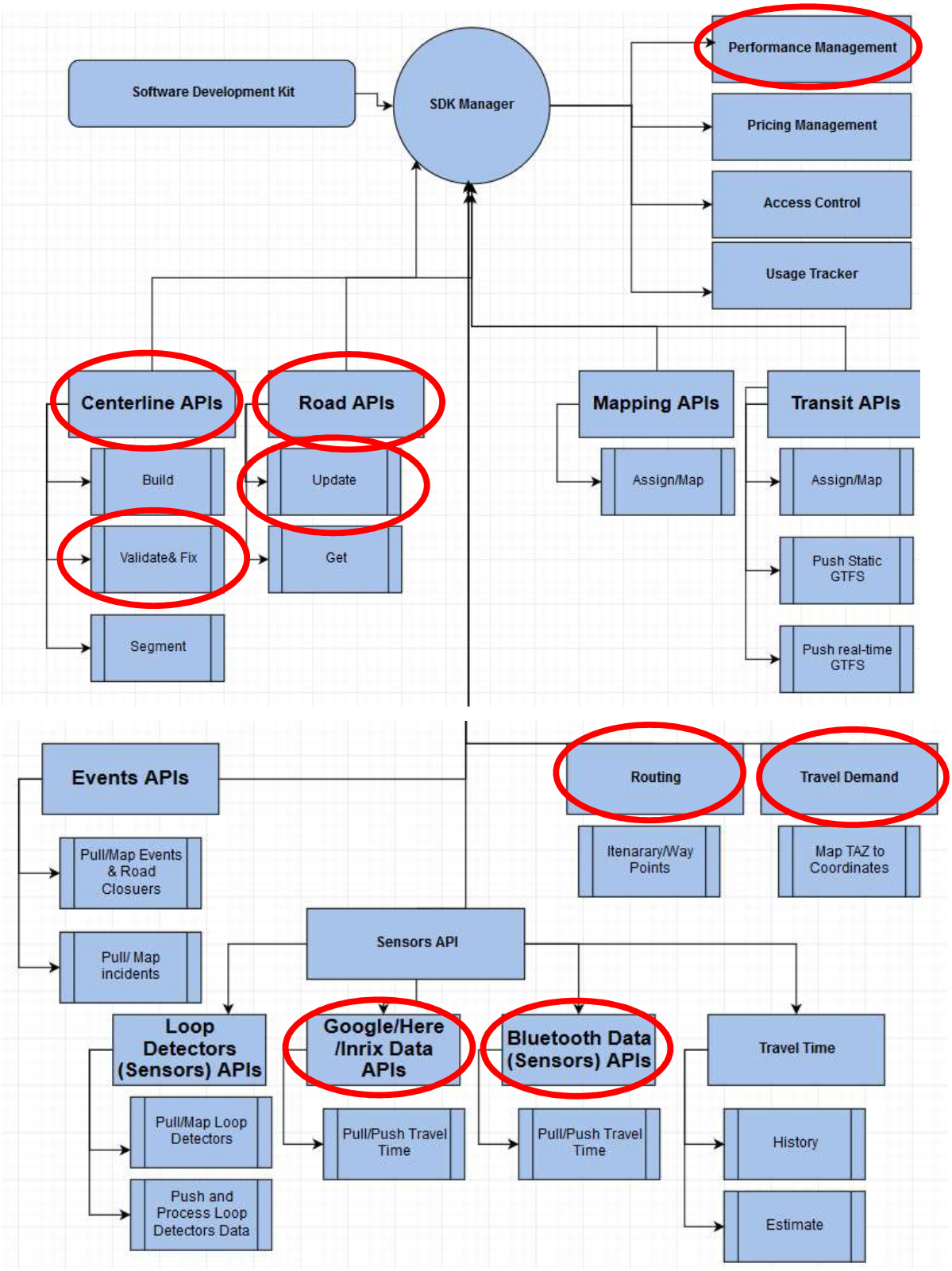

Figure 4-1. Overview of the ATIS SDK

4.1.1 Centreline API

The first set of Web services is the Centreline API. They create a virtual representation of the road network on the database and forms the basis for further research and analysis. This involves abstracting map data from multiple sources into lists of road segments, intersections, and their connectivity. This is done by inspecting the information provided for each road segment and each intersection, then the data relevant to the ATIS SDK are extracted for storage on the database. Relevant data include road name, GPS coordinates, number of lanes, speed limit, etc. This project uses publicly available road inventory data from the municipalities in the Greater Toronto Area (GTA), and the Government of Ontario. In addition, data is also obtained from OpenStreetMap (OSM) [15], a worldwide collaborative mapping project that is open data, free to use by the public.

The data provided by the municipalities are in the Esri shapefile format [16], and the PyShp library was used to convert the data [17]. On the other hand, the data provided by the Government of Ontario are in the Esri geodatabase format, and the ArcMap program was used to extract the data. Lastly, the data available from OSM is in a format specific to OpenStreetMap, and a third-party library, PyOsmium [18], is needed to convert this data into the desired format. Once the desired data is extracted from these sources, they are reformatted and stored according to the relations in the database.

4.1.2 Roads API

The second set of Web services is the Roads API, it allows the roads database to be searched using the road names or identifiers. In addition, road intersections can also be queried by providing multiple road names as inputs. If the identifier of the road element is known, information can be queried directly using this API. However, this has limited use since the identifiers used in the database are unique to the system. Since the goal of the ATIS is to provide travellers with information about the real world, there needs to be a way to query the database using elements such as road names and addresses. Furthermore, another Web service is developed to return all road sections between two intersections. This is particularly useful when during road events such as incidents or closures since only a certain section of a road will be affected. Lastly, any changes

to the road network, such as updating road items with new information, or deleting road items, are also included in this API. These functionalities are provided by searching the road elements in the database by name using SQL (Structured Query Language) queries and return all relevant information to the caller.

4.1.3 Travel Demands API

The third set of Web services is the Travel Demand API, which allows retrieving zonal centroid data from the database. The transit agencies around the Greater Toronto Area divides GTA into multiple zones for the purpose of transportation analysis. Therefore, information about the residents and their travel habits are provided to UTTRI at the zonal level. Given the zonal travelling data, the geographic centroid of each zone is needed in order to perform traffic assignment. This functionality is provided by storing the centroid information of each zone in the database and retrieve them on demand.

4.1.4 Traffic Information API

The last set of Web services is the Traffic Information API, it includes gathering traffic data from external sources. The first Web service in this section involves Bluetooth travel data from Toronto Open Data, which collects travel times information based on Bluetooth sensors. Toronto Open Data provides travel times information for the past 4 years to the public in a comma-separated values (CSV) format, which is then stored in the database using Python. The second Web service involves collecting data from Google's Direction API, which returns travel time predictions between two points at a specified time in the future [19]. The Google Directions API is used to generate traffic predictions for each route in the database, across different times in the morning rush hour period.

4.2 Azure API Management

Azure API Management, the API management solution from Microsoft Corporation, is used during the development of the ATIS SDK. Azure API Management provides a platform for API testing and publishing and allows connection to the backend Web services, which is hosted on a virtual machine on Microsoft Azure. In addition, it allows the developers to "get near real-time usage, performance and health analytics" [12]. Lastly, it provides the research team with a way to restrict and control access to the APIs to effectively utilize the limited resources.

Azure API Management performs these functions by providing a gateway through which the API is exposed to the public, as well as an administrative portal and a developer portal [20]. The API gateway works by first verifying the credentials of the caller, then accepts the API calls and routes them to the backend Web services while logging the metadata for analytics purposes [20]. The administrative portal is where the owner of the APIs can manage user policies, pricing, and quotas, as well as obtain insight from analytics [20]. Finally, the developer portal provides an interface for the end users to review API documentation, try out different APIs, and create accounts to subscribe to the Web services [20].

4.3 SOAP Implementation

The SOAP-based Web services are developed using the Flask extension Flask-Spyne [21]. The Flask-Spyne library is selected because it is built on the Flask framework [21], and it is the most up-to-date Python library that supports implementing SOAP-based Web services at the time conducting this study. With that being said, it still utilizes the outdated SOAP 1.1 standard and Python 2.7 [21]. The Flask-Spyne extension automatically creates a WSDL file for the SOAP-based Web services that can be published on Azure API Management.

4.4 RESTful Implementation

The RESTful Web services are built using Flask-RESTful, an extension to the Flask framework that adds support for building RESTful APIs [22]. After the API is created and the server is set up, the API is published using Azure API Management. Azure API Management requires the use of OpenAPI specification for publishing RESTful APIs [23]. Therefore, a Flask extension, Flasgger [24], is used to extract the OpenAPI specification for the RESTful API.

Since REST is an architectural style rather than a standard, the Flask-RESTful extension can be viewed as an interpretation of a design that adheres to RESTful principles. This thesis treats the extension as a black box and does not investigate the design choices made in the Flask-RESTful implementation.

In both the SOAP and REST implementations, Python 2.7 is used to ensure a fair comparison. In addition, the Flask API server is a virtual machine hosted on Microsoft Azure, the cloud computing service created by Microsoft, with 2 virtual CPUs, 4 GB of memory, and Ubuntu Server 18.04 as its operating system. Furthermore, all data are stored in a back-end SQL database, also hosted on Microsoft Azure. When a request is made to the API portal on Azure API Management, the request is sent to the respective Flask server. The server then processes the request and queries the requested data from the SQL Server using SQL queries. Afterwards, the server sends the response to Azure API Management, which then returns the response to the client. Figure 4-2 illustrates the data flow in the implementation used in this thesis.

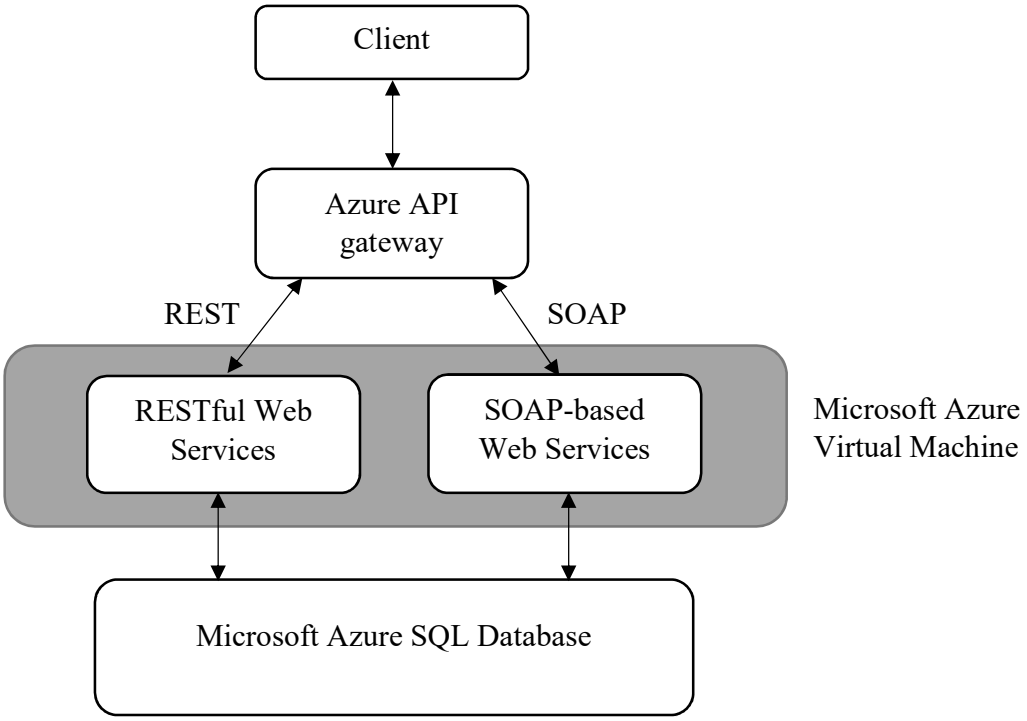

Figure 4-2. Overview of the Web service architecture

4.5 Performance Metrics

Azure API management provides a tool, Azure Monitor, that monitors APIs and includes many of the metrics mentioned previously [1]. Therefore, for the scope of this paper, no additional monitoring tools are developed, and the comparison will be based solely upon the metrics provided by Azure Monitor. In addition, the metrics are recorded at the API gateway level. Azure Monitor does not monitor the CPU and memory utilization directly, thus these metrics will not be included in the tests. In addition, latency measures the delay introduced by any links between the client and the server [9]. Since all the APIs developed during testing will be hosted on the same server, the latency metric is not meaningful and will also be omitted in the tests. Lastly, security-related metrics are beyond the scope of this paper due to the difficulty in establishing meaningful metrics and measuring them. The following table provides a summary of the metrics that will be used in this paper, adapted from [1].

Table 4-1. List of metrics to be measured

Name of metric	Units	Description
Response time	ms	Time elapsed from the moment gateway receives request until the moment response is sent in full
Request size	bytes	The packet size of the request from the client
Response size	bytes	The packet size of the response sent by the server

4.6 Testing Procedure

For this paper, five Web services from the previously mentioned APIs are selected for comparison between SOAP and RESTful. They are chosen for a balanced representation of all the APIs mentioned above, as well as include all four CRUD operations. The five Web services chosen are listed in Table 4-2. Testing is done by repeatedly invoking these Web services and recording the metrics listed above.

Table 4-2. List of Web services to be compared in this paper

Name of Web service	Type	Description
Centreline API: Process OpenStreetMap Data	Create	Given the data file extracted from OpenStreetMap, create a virtual road network in the database
Roads API: Retrieve road item	Read	Given a road item number, find the corresponding road item in the database and return relevant attributes
Roads API: Update road item	Update	Given a road item number and the updated values, update the corresponding road item in the database
Roads API: Delete road item	Delete	Given a road item number, remove the corresponding road from the database
Travel Demands API: Search centroid by zone	Read	Given a zone number, return its geographical centroid

Chapter 5 Results

During testing, each Web services was invoked 1000 times, thus the number of observations for each service is 1000. Where possible, parameters such as road item number or zone number are changed across requests, according to a predefined pattern that is used for both the SOAP-based and RESTful Web services. Once the testing is finished, Azure API Management stores a log of all requests on Microsoft Azure in JSON format. This log is downloaded then analyzed to obtain the results presented in this section. Table 5-1 and Figure 5-1 present the results for request sizes, while Table 5-2 and Figure 5-2 present the results for response size. Lastly, Table 5-3 tabulates the results for response time. The results for response time are not presented in a bar chart due to the large variance observed in the data.

Table 5-1. Request size

	Sample mean (bytes)		Standard deviation (bytes)	
	SOAP	REST	SOAP	REST
Centreline API: Process OpenStreetMap Data	646	254	0	0
Roads API: Retrieve road item	660	243	0	0
Roads API: Update road item	828.9	340.9	0.3	0.3
Roads API: Delete road item	666	246	0	0
Travel Demands API: Search centroid by zone	677.2	237.2	0.4	0.4

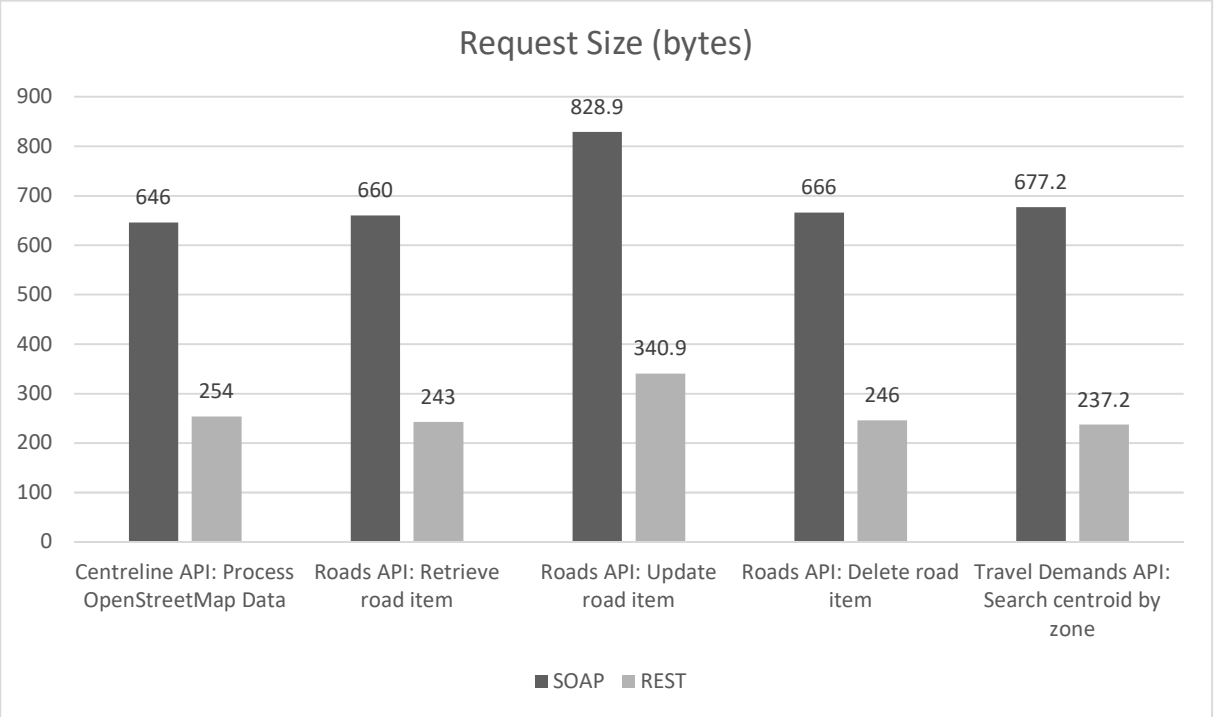

Figure 5-1. Request size comparison

Table 5-2. Response size

	Sample mean (bytes)		Standard deviation (bytes)	
	SOAP	REST	SOAP	REST
Centreline API: Process OpenStreetMap Data	419	149	0	0
Roads API: Retrieve road item	1618.7	1307.5	10.4	5.2
Roads API: Update road item	415	154	0	0
Roads API: Delete road item	415	95	0	0
Travel Demands API: Search centroid by zone	562.2	233.2	0.4	0.4

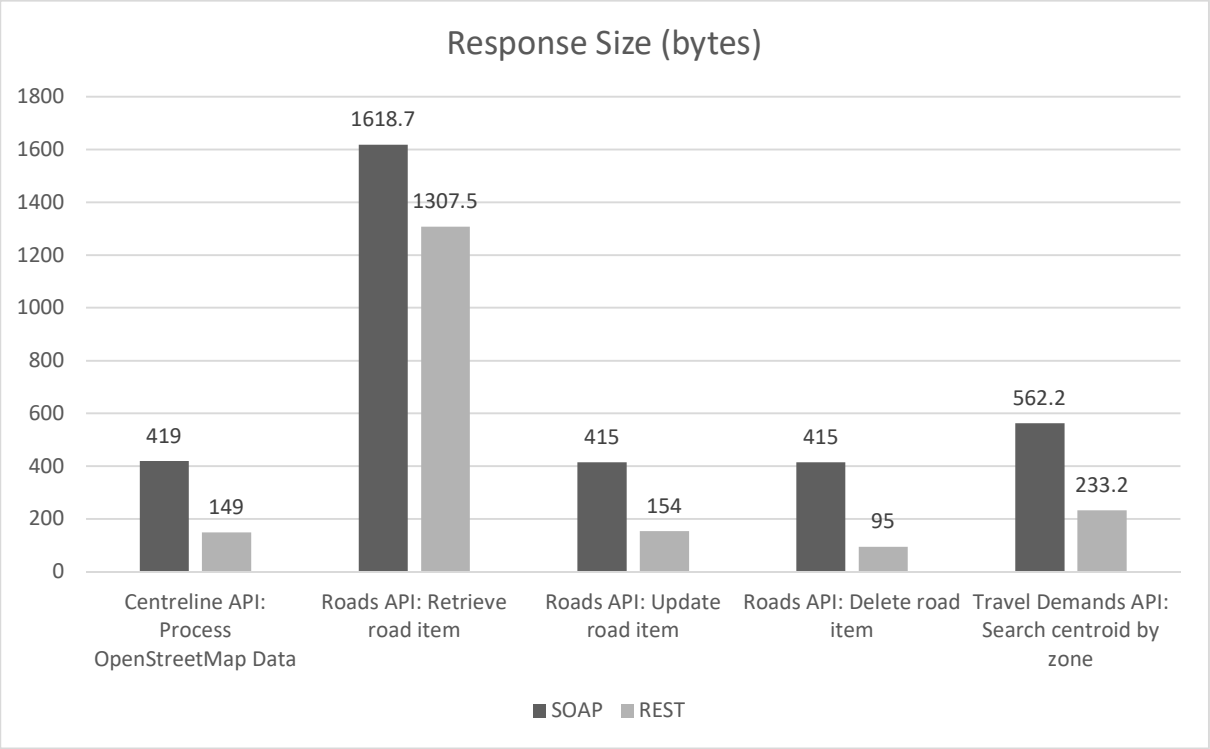

Figure 5-2. Response size comparison

Table 5-3. Response time

	Sample mean (ms)		Standard deviation (ms)	
	SOAP	REST	SOAP	REST
Centreline API: Process OpenStreetMap Data	10029.7	10092.2	1197.3	502.0
Roads API: Retrieve road item	37.5	37.7	19.2	41.8
Roads API: Update road item	45.0	45.1	15.5	22.2
Roads API: Delete road item	46.6	45.4	33.1	36.8
Travel Demands API: Search centroid by zone	33.0	33.2	9.9	22.2

Chapter 6 Discussions

The test results show that the in the SOAP-based Web services have request sizes between 390 and 490 bytes, as well as response sizes between 260 and 330 bytes larger than their RESTful counterparts. This is consistent with the finding of [2] and [3] as presented in Chapter 2. Larger packet sizes translate to increased cost on a cloud computing service due to the increased bandwidth usage. On the other hand, it appears that SOAP-based Web services have a slightly faster mean response time compared to the REST-based Web services, which is inconsistent with all the literature reviewed in Chapter 2. This can be somewhat explained by the large variation of the measured response times. A two-tailed T-test with a p-value of 0.05 shows that the null hypothesis that the two sample distributions are equal cannot be rejected in any of the Web services tested. Therefore, the test result does not draw any conclusive results on the response time comparison and suggests that more work is needed to control the variance of the measured response times.

The large variation in the measured response time may be due to the usage of the cloud platform in the test. Firstly, the developers cannot directly control the resources used on the cloud platform, where the provider may allocate different resources to the server over time depending on the server load. In addition, since the response time is only measured at the Azure API gateway, we cannot determine the compute time at the server and the database, as well as the latency between the gateway, server, and the database. Overall, more work is required to investigate the effects of cloud computing on the measured results.

Chapter 7 Conclusions and Recommendations

In this paper, a comparison between SOAP-based and RESTful Web services is made using a transportation-themed API hosted on Microsoft Azure. The APIs compared by this paper are selected based on the needs of a transportation research group to cover the common database operations and HTTP methods. The results show that RESTful Web services have a smaller network load compared to SOAP-based services, while the response time is roughly the same between the two architectures. In addition, building a SOAP-based server today can entail using outdated standards and specifications that are no longer being maintained, while support for RESTful services are widely available and constantly updated. Therefore, if one were to start building a set of Web services today, there is little advantage to select a SOAP-based approach, and a RESTful implementation is recommended.

References

[1] Microsoft Corporation, "Monitor published APIs," 14 June 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/api-management/api-management-howto-use-azure-monitor. [Accessed 26 October 2018].

[2] G. Mulligan and D. Gracanin, "A Comparison of SOAP and REST Implementations of a Service-Based Interaction Independence Middleware Framework," in *2009 Winter Simulation Conference*, Austin, TX, USA, 2009.

[3] F. Belqasmi, J. Singh, S. Y. Bani Melhem and R. H. Glitho, "SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing," *IEEE Internet Computing,* vol. 16, no. 4, pp. 54-63, 8 May 2012.

[4] K. Mohamed and D. Wijesekera, "Performance Analysis of Web Services on Mobile Devices," *Procedia Computer Science,* no. 10, pp. 744-751, January 2012.

[5] S. Kumari and S. K. Rath, "Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration," in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Kochi, India, 2015.

[6] M. Garriga, C. Mateos, A. Flores, A. Cechich and A. Zunino, "RESTful service composition at a glance: A survey," *Journal of Network and Computer Applications,* pp. 32-53, 2015.

[7] S. Patni, Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS, Santa Clara, California: Apress Media, LLC, 2017.

[8] World Wide Web Consortium, "SOAP Version 1.2 Part 0: Primer (Second Edition)," 27 April 2007. [Online]. Available: https://www.w3.org/TR/2007/REC-soap12-part0-20070427/#L1161. [Accessed 12 October 2018].

[9] B. De, API Management: An Architect's Guide to Developing and Managing APIs for Your, Bangalore, Karnataka, India: Apress Media, 2017.

[10] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, Irvine, CA, 2000.

[11] H. Niu, I. Keivanloo and Y. Zou, "API usage pattern recommendation for software development," *Journal of Systems and Software,* pp. 127-139, 2016.

[12] Microsoft Corporation, "API Management," 12 October 2018. [Online]. Available: https://azure.microsoft.com/en-ca/services/api-management/.

[13] J. Snell, D. Tidwell and P. Kulchenko, Programming Web Services with SOAP, Sebastopol, CA: O'Reilly & Associates, Inc., 2001.

[14] A. Ronacher, "Welcome," Flask (A Python Microframework), 2019. [Online]. Available: http://flask.pocoo.org/. [Accessed 18 March 2019].

[15] OpenStreetMap, "Copyright and License," 2018. [Online]. Available: https://www.openstreetmap.org/copyright. [Accessed 30 October 2018].

[16] Environmental Systems Research Institute, Inc., "ESRI Shapefile Technical Description," July 1998. [Online]. Available: https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf. [Accessed 19 March 2019].

[17] J. Lawhead, "PyShp," 2013. [Online]. Available: https://github.com/GeospatialPython/pyshp. [Accessed 19 March 2019].

[18] S. Hoffmann, "PyOsmium," 2018. [Online]. Available: https://osmcode.org/pyosmium/. [Accessed 19 March 2019].

[19] Google, "Directions API: Developer Guide," 16 January 2019. [Online]. Available: https://developers.google.com/maps/documentation/directions/intro. [Accessed 5 March 2019].

[20] Microsoft Corporation, "What is API Management?," 14 November 2017. [Online]. Available: https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts. [Accessed 1 November 2018].

[21] Spyne Contributors, "Spyne Documentation," [Online]. Available: http://spyne.io/docs/2.10/. [Accessed 5 February 2018].

[22] K. Burke, K. Conroy, R. Horn, F. Stratton and G. Binet, "Flask-RESTful - Flask-RESTful 0.3.6 documentation," 2018. [Online]. Available: https://flask-restful.readthedocs.io/en/latest/. [Accessed 4 December 2018].

[23] OpenAPI Initiative, "Home," OpenAPI Initiative, 2019. [Online]. Available: https://www.openapis.org/. [Accessed 19 March 2019].

[24] B. Rocha, "Flasgger," 2014. [Online]. Available: https://github.com/rochacbruno/flasgger. [Accessed 19 March 2019].