The Islamic University–Gaza
Research and Postgraduate Affairs
Faculty of Information Technology
Master of Information Technology

الجامعـــــــــة الإســــلاميــة – غـزة
شئون البحث العلمي والدراسات العليا
كـليـــــة تكنـــولوجـــيـــا المعلومـــات
ماجستير تكنـــولوجـــيـــا المعلومـــات

Time Estimation for Next Request to Prevent DOS Attack on RESTful Services

حساب الوقت للطلب التالي لمنع عملية حدوث حجب الخدمة في خدمات الريستفول RESTful

Mona Ismail Subaih

Supervised by:

Dr. Tawfiq Barhoom

Associate Professor – Applied Computer Technology

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Information Technology

July / 2016

Abstract

Systems are getting integrated faster and easier using web API, as applications and cloud API's nowadays are shifting to REST-based services in the detriment of SOAP-based ones. RESTful services are a lightweight alternative to Web Services implemented using HTTP and principles of REST. Thus there is no standard applied on RESTful, so security is not considered by default. One of the most attack suffer by the mainstream service providers is Buffer overflow in RESTful services, as a result from misuse or intentional attack. Client requests a resource many times that consume processing time and a lot of money for each request and may cause Denial of services (DOS). This is a hot topic since there is a lack of study in this field and a wide use for RESTful services as a commercial base, so our approach focuses on how to prevent the suspicious repeated RESTful requests.

Every RESTful request has process time and guarded with a token which we increase in live time by next expected time for next request. To protect the service provider from suspicious repeated RESTful requests (which causes losing money & may cause buffer overflow DOS attack) we must prevent repeated request for same resources from the same client before the process time of the previous request is done and to ensure that the new RESTful request has a valid token.

We propose an approach to estimate RESTful process request time from a set of previous requests using large number of experiments to find general equation for estimate current computing time and finding the next expected time for next RESTful request using our equation.

We compute guard time depending on the next request time which protect service provider from repeated request that causes buffer overflow DOS attack. The results were sufficient as the accuracy ranges between 93% and 98% with average 97.31 %.

Keywords: DOS, RESTful, Buffer overflow, JSON, Tokens, Expected time

الملخص

أصبحت خدمات الويب والواجهات السحابية تتحول في الوقت الحاضر للاعتماد على الخدمات المريحة (RESTful Services) بدلا من استخدام بروتوكول سواب (أحد البروتوكولات المستخدمة في عملية نقل البيانات بين شبكات الكمبيوتر) حيث يتم الحصول على نظم متكاملة بشكل أسرع وأسهل باستخدام واجهة برمجة التطبيقات على شبكة الانترنت. ان الخدمات المربحة هي البديل الخفيف لخدمات الويب ويتم تطبيقها باستخدام بروتوكول نقل النص الفائق (HTTP) ومبادئ الخدمة المريحة وبالتالي فإن هذه الخدمات لا تتبع أي معايير موجودة لخدمات الويب مما يؤدي الى وجود الثغرات الأمنية في هذه الخدمات.

ثغرات الفيض (Buffer Overflow) هي أحد أبرز وأقدم الهجمات التي يتعرض لها مزودو خدمات الويب المريحة نتيجة سوء الاستخدام او نتيجة الاختراق المتعمد. يطلب العميل خدمة الويب عدة مرات في كل مرة يستهلك الطلب وقت في المعالجة التي تقوم على شغل الموارد مما يؤدي الى حجب الخدمة (DOS) بالإضافة الى التكلفة المادية لكل طلب. ان هذا الموضوع ذو أهمية بسبب نقص الدراسات في هذا المجال والاستخدام الواسع للخدمات المريحة كقاعدة تجارية. وبالتالي دراستنا تركز على كيفية منع عملية تكرار طلبات الخدمات المريحة المشبوهة.

كل عملية طلب لخدمة الريستفول (RESTful) يكون له وقت معالجة ويتم حمايته من خلال رمز يتم زيادة زمن البقاء له عبر الوقت المتوقع للطلب الثاني. لحماية مزود الخدمة من الطلبات المشبوهة والمتكررة التي تتسبب في الخسائر المادية وربما حجب الخدمة يجب منع الطلب المتكرر لنفس الموارد من نفس العميل قبل ان ينتهي وقت المعالجة للطلب السابق، ولضمان ان الطلب الجديد للخدمة المريحة له رمز صالح.

نقوم هنا باقتراح دراسة لحساب وقت معالجة طلب الخدمة المريحة من خلال مجموعة من الطلبات السابقة مستخدمين عدد كبير من التجارب لإيجاد معادلة عامة لحساب الوقت الحالي للعملية وإيجاد الوقت المتوقع للطلب التالي باستخدام المعادلة. ونقوم بعد ذلك بحساب وقت الحماية والذي بدوره يقوم بحماية المزود من الطلبات المتكررة التي تسبب اختراق حجب الحماية.

عند تقييم النتائج وجدنا أن نسبة دقة المعادلة تتراوح بين 93% الى 98 % مع متوسط 97.31 % وهي نتيجة مقبولة تزداد كلما زاد حجم المعلومات المرسلة من خدمة الويب.

الكلمات المفتاحية :حجب الحماية، خدمات مريحة، ثغرات الفيض، رموز، الوقت المتوقع.

Dedication

To my peacemaker and teacher prophet Mohammed.

To my loving parents for give me support and push always.

To my beloved husband for being responsible and supportive.

To my daughters Eman and Ameer whose missing care while the study.

To my brothers and sister as they all kept me going.

To all my friends, colleagues for their endless support.

Acknowledgement

The completion of this thesis could not have been possible without Allah. Thanks to Allah first and foremost who bestowed me the awareness, perseverance and mercy.

I would like to express my sincere appreciation to my supervisor Dr. Tawfiq Barhoom for his steady guidance, help and relentless support.

In addition, I would like to extend my thanks to the academic staff of the Faculty of Information Technology who helped me during my master’s study and taught me different courses.

Table of Content

List of Tables

List of Figures

List of abbreviations

API	Application Programming Interface
CRU	Create, Read, Update, Delete
DOS	Denial of services
GAE	Google App Engine
HAR	HTTP Archive
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MITM	Man-in-the-Middle
QOS	Quality of service
REST	Representational State Transfer
SOAP	Simple Object Access protocol
TTR	Time-To-Repair
URI	Uniform Resource Identifier
WS-Security	Web Services Security
XSS	Cross-Site Scripting

Chapter 1

Introduction

Chapter 1
Introduction

Nowadays, the world is interested in the field of cloud computing, most systems are getting inter-connected faster, as applications and cloud API's make deep use of RESTful services to expose resources to consumers.

REST or Representational State Transfer architecture is designed for web services. Thus resource implements a standard uniform interface (typically HTTP interface), resources have name and addresses (URIs), While RESTful is supported by the simple CRUD set of operations (Create, Read, Update, Delete). The client receives HTTP status codes as feedback result. Therefore JavaScript Object Notation (JSON) is used for exchanging data, JSON is an open standard format that uses human-readable text to transmit data Objects consisting of attribute–value pairs.

RESTful Web Services are stateless. This means that every request is only dependent on itself, one simply has to examine the request to gather all the details concerning it. Stateless also means that the service is more reliable, because there are less steps where something can go wrong. In distributed services, the lack of state means that there is no overhead to keep the different servers consistent (Backere et al., 2014)

There is no standard applied on RESTful, so security is not considered by default. Many types of attacks form security breaches in RESTful service include data confidentiality and integrity as the attacker try to alter the communication (messages) between the client and the provider, or have unauthorized access to resources. In the interim there are many considerable attacks in RESTful services, as it usually public and have wide range access. The attacks are not totally new. However, they are the known attack in the internet such as Man-in-the-Middle (MITM), Replay attack, Spoofing, Message altering, Cross-Site Scripting XSS.

Denial of service (DOS) is an attempt to make a machine or network resource unavailable to its intended users. A (DOS) attack generally consists of efforts to temporarily or indefinitely interrupt or suspend services of a host connected to the Internet, and it is one of the most security threats in RESTful services.

As of 2014, the frequency of recognized (DOS) attacks had reached an average rate of 28 per hour(Preimesberger, 2014) . The main problem here is how to prevent DOS Buffer overflow attack which is created by repeating a valid request multi-time or slow connection attack which causes both consuming resources and long processing time and may end of DOS.

An access token is a string representing an access authorization issued to the client, rather than using the resource owner's credentials directly. Tokens are issued to clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server (Jones, 2012) . The common effective security that all researches agree is security tokens. The token has a limited time albeit it ended the token are not valid anymore and the client need to request new token from provider. The mechanism to extend the time is the computing activity as each time the resource are requested there is constant increment to the computed time. Our approach handle client request tokens refresh time according to expected time for next RESTful request.

Elkurd & Barhoom (2015) propose an algorithm to prevent buffer overflow DOS attack in RESTful services by comparing the difference between current time for RESTful request and the time for previous RESTful request with the current process computing time, then increase the token live time for request depending on the expected time for next request.

Unfortunately, this solution assumes the current process computing time for RESTful request and expected time for next request as a constant. This will represent two problems, the first one: is to assume the process computing time as a constant because the difference between current request time and previous request time might take more than a constant or less than it, if the services takes less than assumed constant the approach will reject the next request which causes a noise for the client and might lose him. On the other side if the difference time takes more than a constant this will give a chance of suspicious repeated requests which lead to overload on service resources and additional cost on service provider. The second problem assumes the expected time for next RESTful request as a constant because our approach depend on this time to increase the token life time which considered as a security breach when the token time increases more than real need.

Our goal is to prevent DOS attack on RESTful services by finding the current process computing time using a number of experiments and analyze the results to get equation then estimate the expected time for next RESTful request in order to give the token a new defined time according to the requested resource and to prevent suspicious repeating requests for each client.

1.1 Problem Statement

RESTful services are subject to greater and greater levels and types of attacks, as hackers exploit vulnerabilities within software, since security is not considered by default and there is no standard can be applied. Denial of the service (DOS) is one of the most security threats in RESTful services, whether it is created intentionally by attacker or by misuse.

The main problem is the assumption of setting the request process computing time with the expected time for next RESTful request as constants, this will give a chance to repeat malicious request on RESTful services which will lead to buffer overflow DOS.

1.2 Objectives

1.2.1 Main Objective

To develop an approach for estimating the process computing time (service time) and the next request time to prevent DOS buffer overflow attack on RESTful Services.

1.2.2 Specific Objectives

The specific objectives extracted from the main objective which are:

1. To collect data and prepare as JSON format to retrieve from RESTful services.
2. To design and Implement RESTful services, and setup on Windows Azure as Cloud environment.
3. To design client script to send large number of requests for RESTful services and capture the time of each request.
4. To Analyze the current computing time for RESTful requests which retrieve different size of data, then get the equation for finding the next request time equation
5. Implement and enhance the algorithm that also includes equation for finding the next request time for RESTful Services.

6. Evaluate the proposed approach by comparing the response time of our experiments with the real response time after applying the equation on our approach.

1.3 Importance of the research

1. Recently, RESTful Services is a hot topic and there is no much researches on it. In general, the web services are based on security standard such as WS-Security but RESTful Services is commercially widespread and that's why there is a security breach which is not taken into account and need more research.

2. Denial of Service (DOS) is considered as one of the most security threats in RESTful services, attackers may create it deliberately, or it can be created by misuse.

3. From the practical side, this topic is important as the current work in my company is based on software production as a service and most of other web services companies face the same issue in how to protect services or how to reduce the consumption of resources to the service which also causes money loss.

1.4 Scope and Limitation

The work is applied with some limitations and assumptions such as:

- The RESTful service may be public and clients are unknown, or private. The research will focus on the private customers of services but not for public services.

- Our experiments are limited to windows Azure environment.

- The results will be evaluated manually, as there is a lack of tools supporting such evaluation.

- The main concern in this research is to detect the current process computing time and expected time for next RESTful request to help in preventing Denial of service for

RESTful services. The research will focus on the accuracy of detecting times while other issues such as accuracy of preventing attack 100 % is not considered.

1.5 Research Methodology

To achieve our objectives will follow a research methodology that consists of the following stages:

- **Research and survey:** this include reviewing the recent literature closely related to the thesis problem. After analyzing the existing methods, identifying the drawbacks of existing approaches.

- **Building the Approach:** the structure of our proposed approach include the following steps:

  1. Gathering dummy data from Cloud API to let the Restful service return it back.

  2. Prepare the collected data set into proper format and different sizes.

  3. Design RESTful services and setup on windows Azure after preparing it.

  4. Develop script using JavaScript to request (GET, POST) RESTful service for several times, then store the request with time process for every request and size of load data.

  5. Analyze times and get the equation for next request time

  6. Design our approach to prevent DOS attack on RESTful services by using the equation to estimate next request time and apply it to our RESTful service to evaluate our results.

- **Implementation:** we implement the preparation phase using PHP and JavaScript to collect the data and transfer it for JSON format with different sizes, and we implement the client side script using JavaScript to send several RESTful requests, also we implement RESTful services using PHP language. Finally, we implement our approach to prevent DOS attack on RESTful services.

- **Evaluation:** To evaluate our approach we will apply several experiments using repeated request application to measure the response time and performance with accuracy for RESTful services.

- **Results and discussions:** in this stage we will analyze the obtained results and justify our approach.

1.6 Thesis Format

The thesis is organized as follows: Chapter 2 covers the theoretical and technical background related to web services, cloud computing and all tools used in our thesis, Chapter 3 presents related work in preventing DOS attacks for web services and QOS of web services, Chapter 4 presents the proposed approach for estimating the next request time of RESTful request, Chapter 5 describes the implementation, Chapter 6 describes the experiments and results. Finally, Chapter 7 concludes the thesis and suggests some future work.

Chapter 2

Theoretical Background

Chapter 2
Theoretical Background

Nowadays, the world is fully interested in building software as services since the WEB services have been raising in recent years and are by now one of the most popular techniques for building distributed systems. Most systems are getting inside-connected faster, as applications and cloud API's make deeply use of web services to expose resources to consumers.

2.1 Cloud Computing

The principal idea of cloud computing was proclaimed route in 1960 by Professor John McCarthy, as; "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility. The computer utility could become the basis of a new and important industry" (Sajid, 2013).

Cloud Computing is the utilization of Internet-based technologies for the provision of services (Marinos & Briscoe, 2009), beginning from the cloud as a metaphor for the Internet, relying on depictions in computer network diagrams to abstract the complex infrastructure it hides (Marinos & Briscoe, 2009). It offers the illusion of unlimited computing resources accessible on demand, with the disposal of forthright duty from users, and payment for the use of computing resources on a short-term basis as needed (Armbrust et al., 2009). In addition , it does not require the node providing a service to be present once its service is deployed (Armbrust et al., 2009). Cloud Computing theoretically consolidates Software-as-a-Service (SaaS) (Road & Kingdom, 2008). Web 2.0 and other technologies with reliance on the Internet, providing common business applications online through web browsers to satisfy the computing needs of users, while the software and data are stored on the servers. Figure 1.1 demonstrates the typical configuration of Cloud Computing at run-time when customers visit an application served by the central Cloud, which is housed in one or more data centers (Buyya, Yeo, & Venugopal, 2008). Green symbolizes asset utilization, and yellow resource provision. Red color is assigned to the role of coordinator for resource provision, which is centrally controlled, even if the central node is implemented as a distributed grid, which is the usual incarnation of a data center, control is still centralized. Suppliers, who are the controllers, are typically organizations with other web activities that require big computing resources, and in their endeavors to scale their essential business have gained impressive skill and hardware.

For them, Cloud Computing is a way to resell these as a new product while expanding into a new market. Consumers include everyday users, Small and Medium Sized Enterprises (SMEs), and ambitious start-ups whose innovation potentially threatens the incumbent providers.

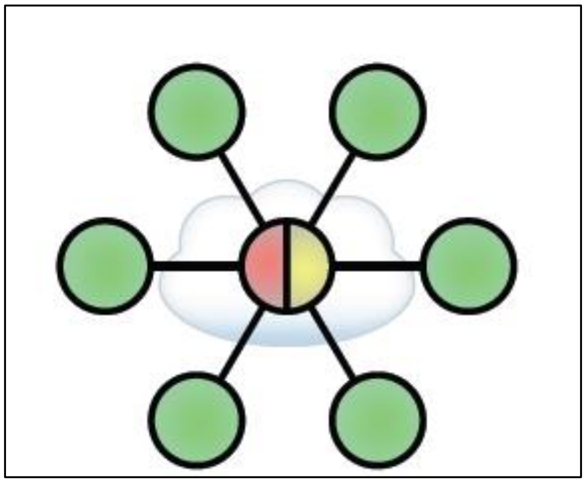


Figure (2.1) Cloud Computing Concepts of work

Cloud computing can help to improve business performance by making a contribution to control the cost of delivering IT resources to any organization. It minimizes the overhead of buying, managing and controlling IT resources. The financial model applied in cloud computing is “Pay-per-Use” so the consumer only pays for his needs.

2.1.1 Cloud Service Model

The principle of cloud computing is to deliver different computing services on Internet which is available as subscription-based services in a pay-per-use model to consumers. These services are essentially categorized under three classes as hierarchy as can be seen in Figure 2.2.

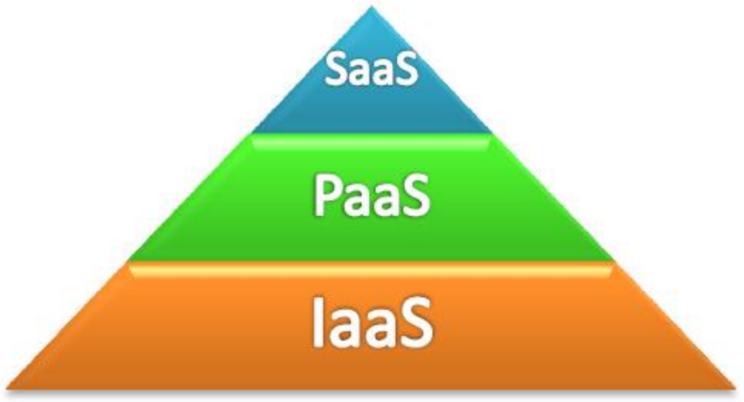


Figure (2.2) Services Provided by Cloud Computing

1. **Software as a Service (SaaS):**

   In this highest level, different types of applications running on cloud environment are provided to the customer. The user can access those applications from various devices through a thin client interface such as a web browser (Sarna, 2015). For example, Gmail is a SaaS where Google is the provider and we are consumers (Nandgaonkar & Raut, 2014).

2. **Platform as a service (PaaS):**

   This intermediate level provide a platform for developers to deploy there applications which are built using programming languages and tools supported by the provider (Sarna, 2015) such as: Google App Engine (GAE), Microsoft Azure, IBM Smart Cloud, Amazon EC2, and salesforce.com (Nandgaonkar & Raut, 2014).

3. **Infrastructure as a Service (IaaS):**

   The last level provides a fundamental computing resources such as processors, storage, and resources (Sarna, 2015). For example: Storage services provided by AmazonS3, and Amazon EBS; and Computation services provided by AmazonEC2, and Layered tech (Nandgaonkar & Raut, 2014).

2.2 DOS attack in APIs

A denial-of-service (DoS) or distributed denial-of-service (DDoS) attack is an attempt to make a machine or network resource unavailable to its intended users, it consists of efforts to temporarily or indefinitely interrupt or suspend services of a host connected to the Internet.

DoS attack, uses many devices and multiple Internet connections, often distributed globally into what is referred to as a botnet.

A DoS attack is, therefore, much harder to deflect, simply because there is no single attacker to defend from, as the targeted resource will be flooded with requests from many hundreds and thousands of multiple sources.

As for Web API's attacks there is many attacks can be classified as Dos Attack such as

- Buffer overflows in the application functions.
- Malformed data to raise unexpected exceptions.
- Exploited race conditions in multi-threaded systems.
- Heavy-duty SQL queries via web forms and "spamming" them with requests, e.g., inserting % characters within search query fields.
- SQL Injection attacks executing recursive CPU-intensive queries.
- The end users' web browsers to overload the application with parallel requests via persistent / reflected Cross-Site Scripting attacks.
- Overly-complex regular expressions within search queries.
- Excessively large files uploaded to the server.

In our thesis we are going to investigate in the effective security manners to reduce DoS attacks in RESTful services.

2.3 Windows Azure Platform

"Windows Azure can be anything you want it to be" (Tulloch, 2013) . As a cloud platform from Microsoft that provides a wide range of different services, Windows Azure lets you build, deploy, and manage solutions for almost any purpose you can imagine. In other words, Windows Azure is a world of unlimited possibilities. Whether you're a large enterprise spanning several continents that needs to run server workloads, or a small business that wants a website that has a global presence, Windows Azure can provide a platform for building applications that can leverage the cloud to meet the needs of your business. (Tulloch, 2013).

2.3.1 Azure Websites

Azure Websites is defined as managed cloud service that permits you to deploy a web application and make it accessible to your clients on the Internet in a very short time. You don't directly support the VMs on which your website runs; they are managed for you (Collier & Shahan, 2015).

Supported languages include .NET, Java, PHP, Node.js, and Python. In addition to creating your own website, there are several web applications available to use as a starting point, such as WordPress, Umbraco, Joomla, and Drupal (Collier & Shahan, 2015).

2.3.1.1 Creating a new website (Collier & Shahan, 2015)

- Start by logging into the Microsoft Azure Preview Portal (portal.Azure.com). At this point, you need an Azure account. If you don't have one, you can sign up for a free trial at Azure.microsoft.com.

- **Using the portal:** after logging into the portal, click the big +NEW icon in the lower-left corner of the screen and select Website, as displayed in Figure 2.4.

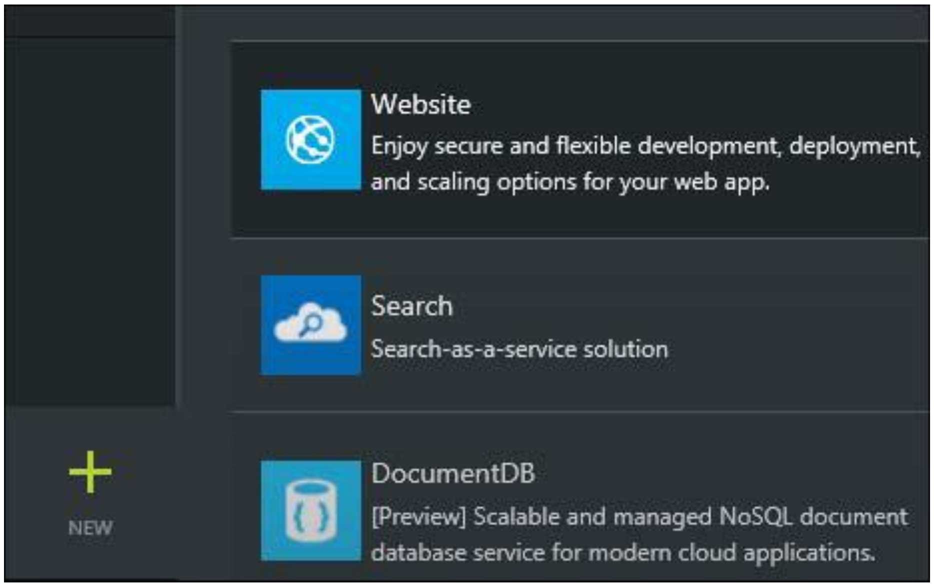
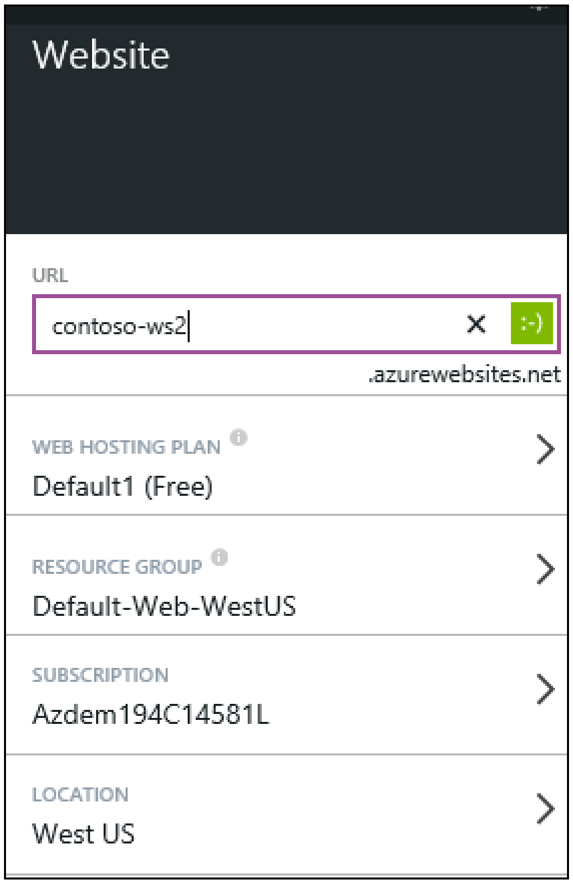

Website
Enjoy secure and flexible development, deployment, and scaling options for your web app.
Search
Search-as-a-service solution
DocumentDB
[Preview] Scalable and managed NoSQL document database service for modern cloud applications.

Figure (2.3) windows Azure portal

You should now see something similar to Figure 2.4, with the fields ready to be filled in.

- The URL must be unique among all of the entries used in Azure Websites. If accepted, there will be a green square with a smiley face in it. Note that whatever prefix is provided here will be appended with. Azurewebsites.net to create the URL for the website.
- SUBSCRIPTION shows the name of the subscription assigned to the Microsoft account with which you logged in. If you administer multiple accounts with the same Microsoft account, you can click SUBSCRIPTION and select the subscription you want to use. LOCATION is the region of the datacenter where the website will be hosted. Select the LOCATION closest to you. Accept the default for RESOURCE GROUP.


Website

URL

contoso-ws2

.azurewebsites.net

WEB HOSTING PLAN

Default1 (Free)

RESOURCE GROUP

Default-Web-WestUS

SUBSCRIPTION

Azdem194C14581L

LOCATION

West US

Figure (2.4) create a new website

- WEB HOSTING PLAN defines the allocation of resources for the website, such as number of cores and memory, amount of local storage, and the features available, such as auto scaling and backups.

- After creating the website, we can transfer our websites files using FileZilla program for FTP transfer

2.3.1.2 Transfer website Using FileZilla (Chepaitis, 2003)

1. In the File menu, choose Site Manager
2. In the next window, click on New Site (bottom left) as shown in figure 2.6

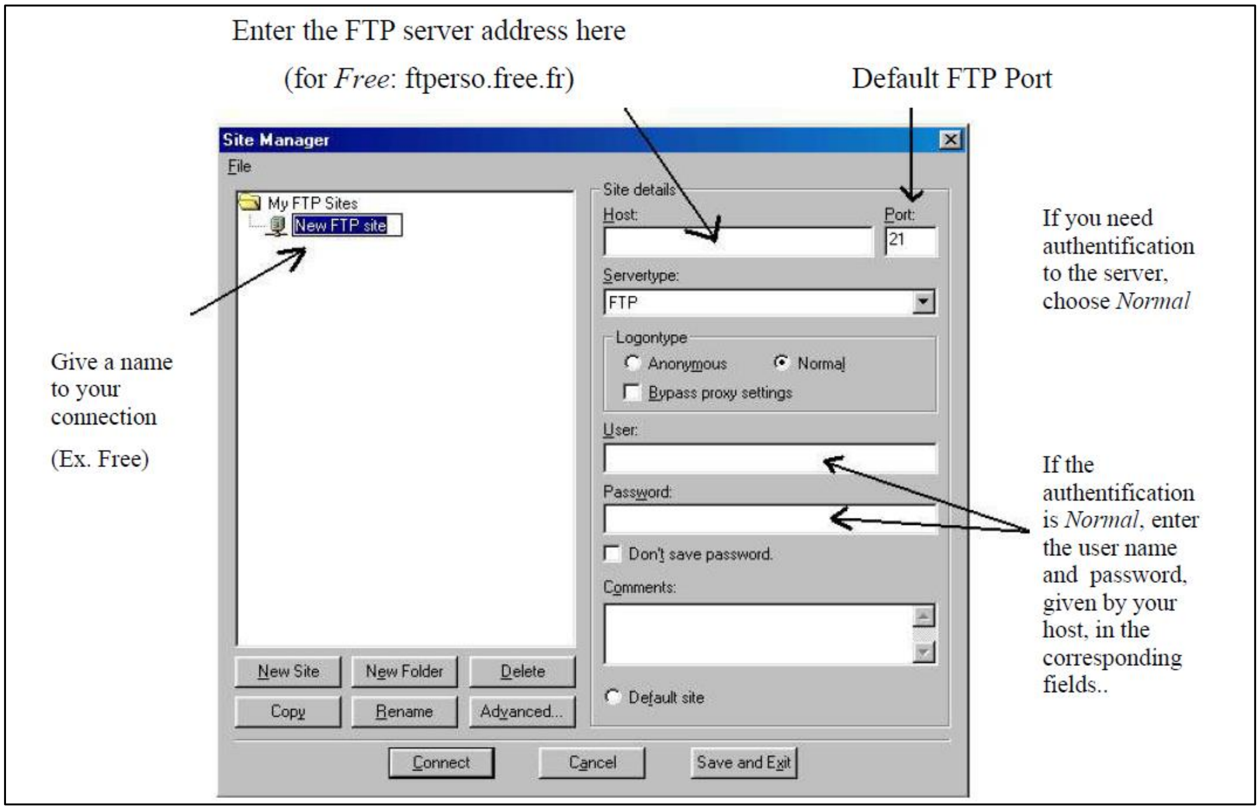


Figure (2.5) create a new size using FileZilla.

3. In order to make a connection to the FTP server, click on the arrow on the right side of the icon the toolbar, and choose a connection name.

4. After connecting to the FTP server, select in the Local Site, one or more files (by pressing the Ctrl key) that you wish to transfer, then drag and drop the file(s) into the Remote Site as shown in Figure 2.6.

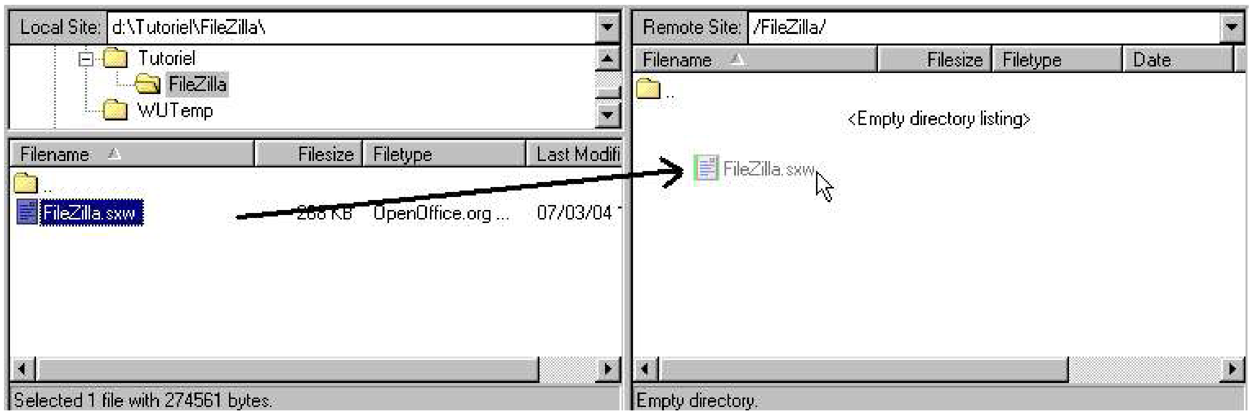


Figure (2.6) upload our websites using FileZilla.

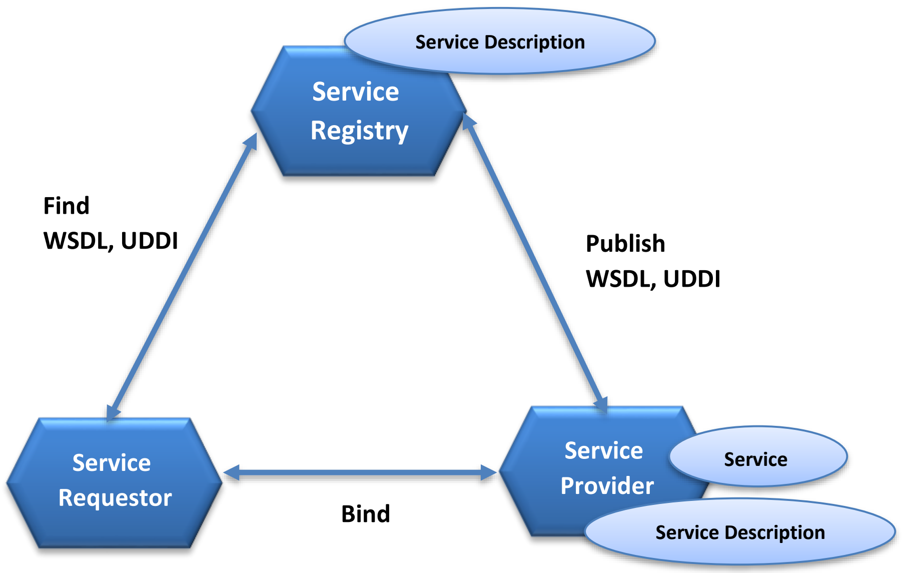
2.4 Web Services Overview

2.4.1 Web Services Definition

Web Services is defined as an interface that describes a collection of operations that are network accessible through standardized XML messaging (Services & Architecture, 2001). Each service has its own description that presented using Extensible Markup Language (XML) notation and known as Service Description. This description covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The communication with Web Service is done through its interface, which conceals the implementation details of the service; this will allow it to be used autonomously of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This gives the Web Services-based applications the advantages to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfill a specific task or a set of tasks. They can be used alone or with other Web Services to do a complex aggregation or a business exchange (Services & Architecture, 2001) .

2.4.2 Web Services Model

Web service interactions based on three main roles: service provider, service registry and service requestor communicate between each other by three main operations publish, find and bind operations as presented in Figure 2.8. Together, these roles and operations act upon the Web Services artifacts: The Web service software module and its description. In a typical scenario, a service provider hosts a network-accessible software module (an implementation of a Web service). The service provider defines a service description for the Web service and publishes it to a service requestor or service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry and uses the service description to bind with the service provider and invoke or interact with the Web service implementation. Service provider and service requestor roles are logical constructs and a service can exhibit characteristics of both (Services & Architecture, 2001) .



Figure(2.7) Web Services roles, operations and artifacts (Services & Architecture, 2001)

2.5 REST Overview

2.5.1 REST Definition

REST (REpresentational State Transfer) is a simple stateless architecture that generally runs over HTTP (Richardson & Ruby, 2008). REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP- and WSDL-based interface design because it's a considerably simpler style to use (Rodriguez, 2015).

REST asks developers to use HTTP methods explicitly and in a way that's consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping between creates, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping (Rodriguez, 2015):

- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.

- To remove or delete a resource, use DELETE.

The formal REST constraints are:

- **Client–server**

In fact, a uniform interface creates a separation area between clients and servers as clients are not concerned with data storage and it remains internal to each server. On the other hand, servers are not concerned with user interface, so that servers can be more scalable.

- **Stateless**

The client–server communication is further compelled by no client context being stored on the server between requests. Each request from any client contains all the information required to service the request, and session state is held in the client. The session state can be transferred by the server to another service such as a database to keep a persistent state for a period and allow authentication. The client starts sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used the next time the client chooses to initiate a new state-transition (Fielding, 2015).

- **Cacheable**

As on the World Wide Web, clients can cache (reserve) responses. Responses should along these lines, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients from reusing stale or unseemly information in response to further demands. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance (Pautasso, Alarcon, & Wilde, 2014).

- **Uniform interface**

The uniform interface constraint is major to the design of any REST service. The uniform interface simplifies and decouples the architecture, which enables each part to evolve independently. The four constraints for this uniform interface are (Pautasso et al., 2014) :

  - Identification of resources

- Manipulation of resources through these representations
- Self-descriptive messages
- Hypermedia as the engine of application state.

2.5.2 Comparison between SOAP and REST

Both SOAP (Simple Object Access Protocol) and REST (REpresentational State Transfer) are methods of communication between two applications that allow access to data. SOAP is a protocol specification for exchanging structured information in computer networks. REST is the architectural style of the World Wide Web, and tries to simplify the method of requesting information. The comparison of two frameworks discussed in table 2.1.

Table (2.1) Comparison of SOAP and REST based Web Services (Pautasso et al., 2014)

SOAP	**REST**
It is well known old traditional Technology.	It is new technology as compared to SOAP
Within the enterprise and in B2B scenarios, SOAP is still very attractive.	This is not to say that REST is not enterprise ready. In fact, there are known successful RESTful implementations in mission critical applications such as banking.
It requires binary attachment parsing.	It supports all data types directly.
SOAP web services always return XML data.	While REST web services provide flexibility in regards to the type of data returned.

SOAP	REST
It consumes more bandwidth because a SOAP response could require more than 10 times as many bytes as compared to REST.	It consumes less bandwidth because it’s response is lightweight.
SOAP request uses POST and require a complex XML request to be created which makes response-caching difficult.	Restful APIs can be consumed using simple GET requests, intermediate proxy servers / reverse-proxies can cache their response very easily.
SOAP uses HTTP based APIs refer to APIs that are exposed as one or more HTTP URIs and typical responses are in XML / JSON. Response schemas are custom per object	REST on the other hand adds an element of using standardized URIs, and also giving importance to the HTTP verb used (i.e. GET / POST / PUT etc.)
False assumption: SOAP is more secure. SOAP use WS-Security. WS-Security was created because the SOAP specification was transport-independent and no assumptions could be made about the security available on the transport layer.	REST assumes that the transport will be HTTP (or HTTPS) and the security mechanisms that are built-in to the protocol will be available
Is the prevailing standard for web services, and hence has better support from other standards (WSDL, WS) and tooling from vendors.	Lack of standards support for security, policy, reliable messaging, etc., so services that have more sophisticated requirements are harder to develop.

2.5.3 RESTful Services Usage Statistics

According to some statistics from Programmable Web (Avram, 2011), an API resource indexer, showing that SOAP has grown over the years, but it has a smaller share than REST, which has had consistent growth:

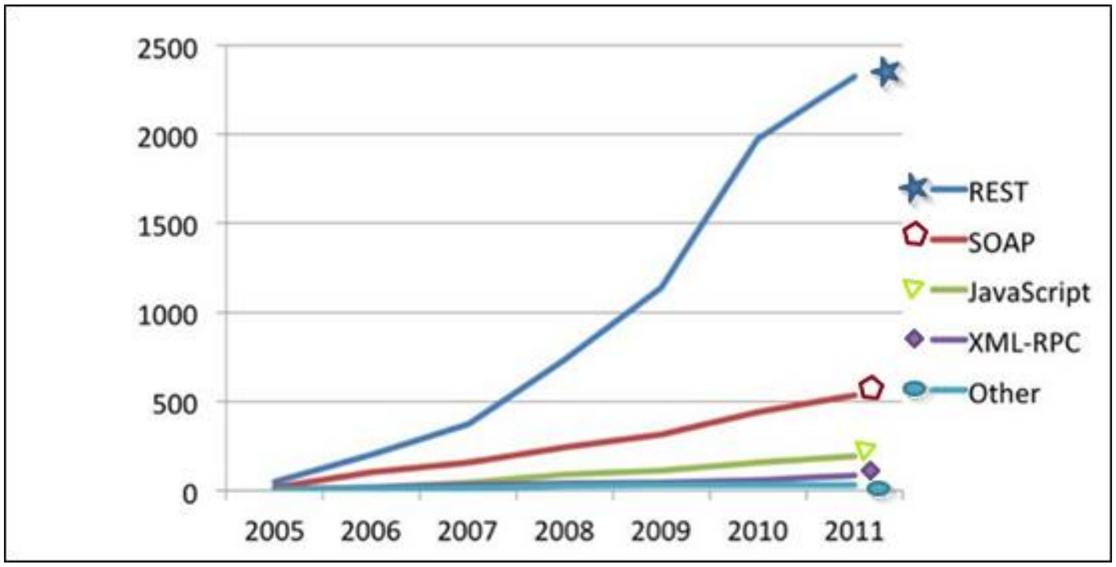


Figure (2.8) REST vs. SOAP Chart

Figure 2.9 shows the high usage of Rest which reaches 73% of APIs usage and this is shows the large scale of Rest usage in enterprise applications. From this point most of service providers are looking for protecting their services from DOS attacks, and this makes protecting Rest Service a very hot topic.

2.6 Mashape Cloud API

Mashape is considered as a cloud API Management and Marketplace where a developer can find, circulate, and expend public and private APIs in addition to offering free or monetized APIs (Team, 2015). To start this, sign up for a new account then log in. You will have with the following:

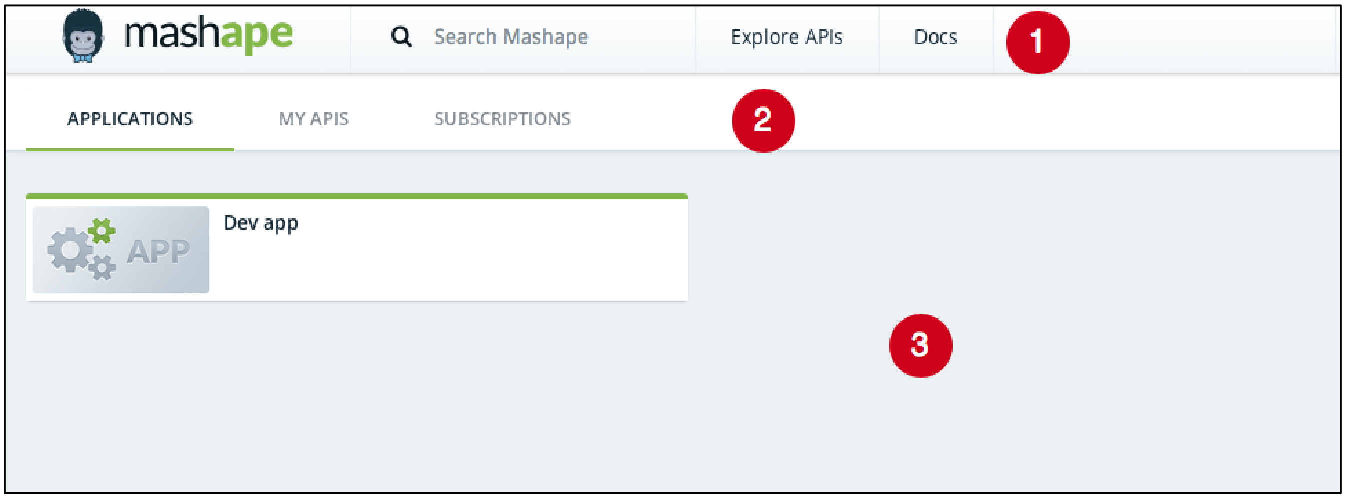


Figure (2.9) Mashape's Dashboard

1. **Site navigation** (Team, 2015)
   - Search Mashape: Find an API in the marketplace by name or keyword
   - Explore APIs: Explore the marketplace for public APIs
   - Docs: Developer documentation, guides and tutorials
   - Applications: easy and quick access to all the applications you've created on Mashape
   - My APIs: Quick access to all the APIs you've added to Mashape
   - Your Profile: Ability to Access to your profile settings
2. **Applications Navigation** (Team, 2015)
   - Applications: A dashboard showing all the APIs you've consumed in applications, with graphs of analytics over time.
   - My APIs: A dashboard showing all the APIs you've added to Mashape, showing graphs of analytics over time.
   - Subscriptions: A list of all the APIs you've consumed on Mashape, FREE and PAID, best way to review Pricing of APIs you're associated with
   - Filter: Type the name of the application to filter the analytics.
   - Analytics Period: Shows analytics over a certain period of time.

3. **Applications Dashboard** (Team, 2015)

Applications are a way to group APIs together. Supposed you consume different APIs for various applications, you might be interested in how the APIs perform. This is where "Applications" come in handy.

2.6.1 Create an application

When you consume an API on Mashape you assign it to an Application, by default this is the Default Application (Team, 2015).

All the APIs you've consumed on Mashape are listed below, grouped by the Application they are assigned to. You can quickly see the status of APIs in a glance.

There are two ways to create an application:

- From the site navigation toolbar
- From the dashboard clicking the Add application button

After specifying the name of the application (I picked "New App") you will be greeted with the following screen.

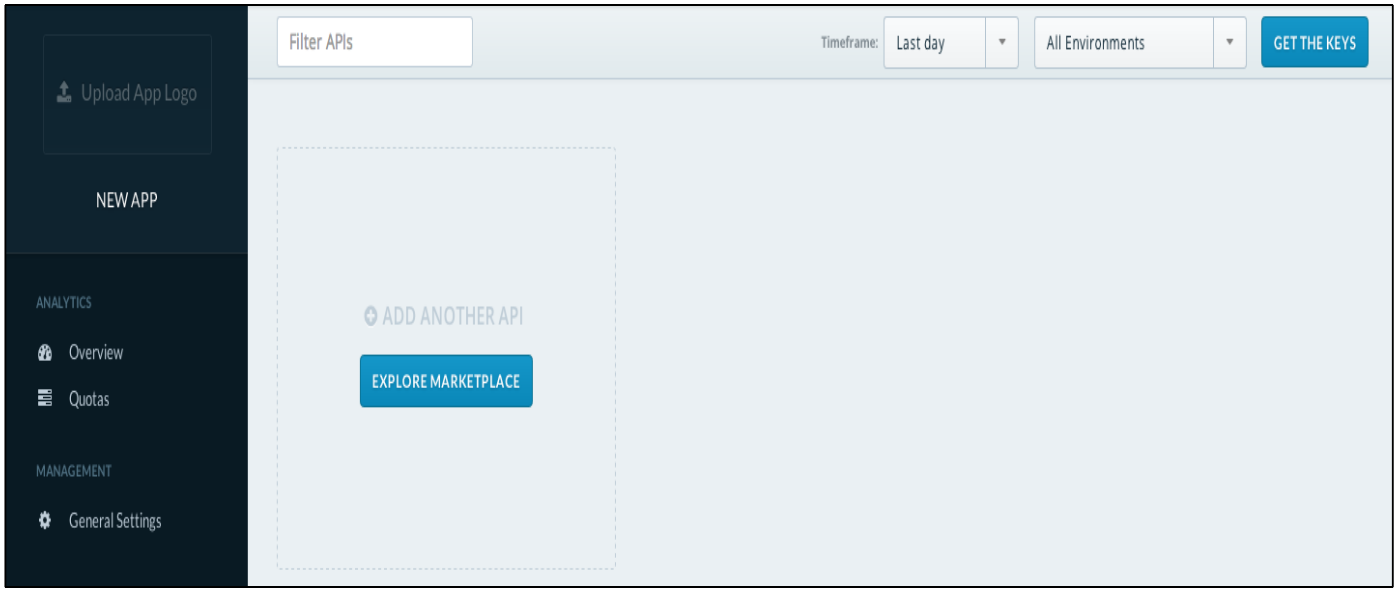


Figure (2.10) Mashape's new application

- **On the left sidebar:**
  - Upload App Logo: To help you identify your apps easily, you can pick and upload an app logo from your computer.
  - Overview: Summary of the APIs contained within this application.
  - Quotas: Number of requests (directly related to the period displayed in the dropdown menu on the top right)
  - General Settings: This view allows you to specify or rename your application, description and to delete the application.
- **On the top menu of this view:**
  - Filter: Use the filter to select specific APIs to view its analytics within the application
  - Analytics Dropdown: Filtering by period and environment for analytics purposes
  - Get The Keys: Retrieve your Development or Production keys, used as authorization headers in all your API calls to Mashape when consuming one of the APIs from the marketplace. You can re-generate (and block previous ones) keys in case you misplace them or if you believe they might be compromised.

In our thesis we use our application called dummyDataApps this application connects with several API to get dummy data to use it in our RESTful services.

2.7 Web Service Quality of Service (QoS)

(QoS)-oriented systems are very dependent on the quality of employed web services. As Web services basically accessed through Internet, achieving high quality of web services is becoming more and more essential (Zheng, Zhang, & Lyu, 2014). In this section, we have a study about (QoS) to ensure that it will not affect of nonfunctional characteristics of web services such as response time, throughput, failure probability, availability, price, and popularity.

- **Web service QoS requirements**

The major requirements for supporting QoS in Web services are as follows:

1) **Availability**: Availability is the quality aspect of whether the Web Service is present or prepared for prompt use. Availability represents the probability that a service is available (Araban, 2004) . In our approach we are keen to keep RESTful services is available all the time by preventing the DOS attacks on it.

2) **Accessibility:** Accessibility is the quality aspect of a service that represents the degree it is capable of serving a Web service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time. There could be situations when a Web service is available but not accessible. High accessibility of Web services can be achieved by building highly scalable systems (Araban, 2004). By preventing buffer overflow on RESTful service we ensure that our services are always accessible.
3) **Scalability:** Scalability refers to the ability to reliably serve the requests despite variations in the volume of requests (Saeed Araban, 2004) .

4) **Performance:** Performance is part of quality of Web service, which is measured in terms of throughput and latency. Good performance of a Web service means the service has higher throughput and lower latency values (Zheng et al., 2014) . In our approach we aim that the response time of the RESTful service will not affected.

5) **Security:** is the quality part of the Web service of giving secrecy and non-denial by verifying the parties included, encoding messages, and giving access control. Security has included significance since Web service invocation happens over the public internet The service supplier can have diverse methodologies and levels of giving security relying upon the service requestor (Araban, 2004) . The main goal of our approach is preventing the RESTful service from DOS attack.

2.8 Overview of Response Time for Web Service

In this section, we have a study about response time to ensure that it will not affect the quality of service during the design of our Restful service and our approach.

In case that application has SOA design standards, is intensely depends on a 3rd party service provider, then user will be baffled sooner or later when the application become slow or crashes. The issue is that: the end user experience and quality of service (QoS) is just comparable to the QoS of the service provider. Along these lines, unless you monitor QoS you cannot measure QoS–and on the off chance that you can't quantify QoS, you can't deal with your service provider and your end user experience. For instance, see this client e-commerce application which has 7 JVM's, 1 database and 7 outside web service providers (Mappic, 2012).

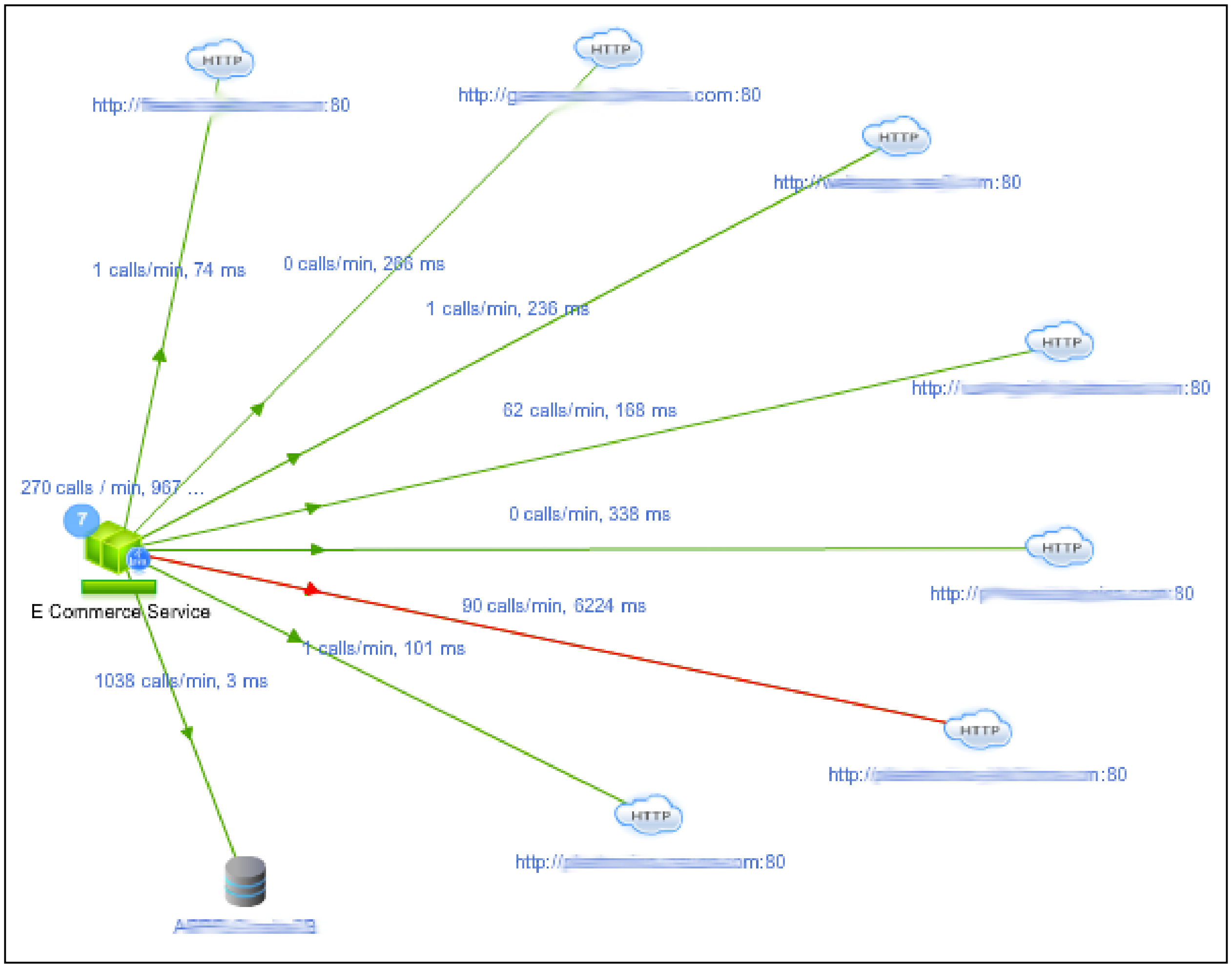


Figure (2.11) E-commerce application (Mappic, 2012)

Figure 2.12 presents the service time that have drop or have big time in red arrow, as the service need to be faster than it. The following is the response time of another web service (PayPal) for a client application over the most recent 3 months. Observe the spikes in response time and take a gander at deviation between average & maximum response time over the time period. What's great is that in spite of the incidental service blip, the PayPal service has gradually enhanced by 14% from 450 milliseconds to around 385 milliseconds. It's additionally been extremely steady in the most recent weeks, alongside having a predictable service (little deviation from average and maximum response time) (Mappic, 2012).

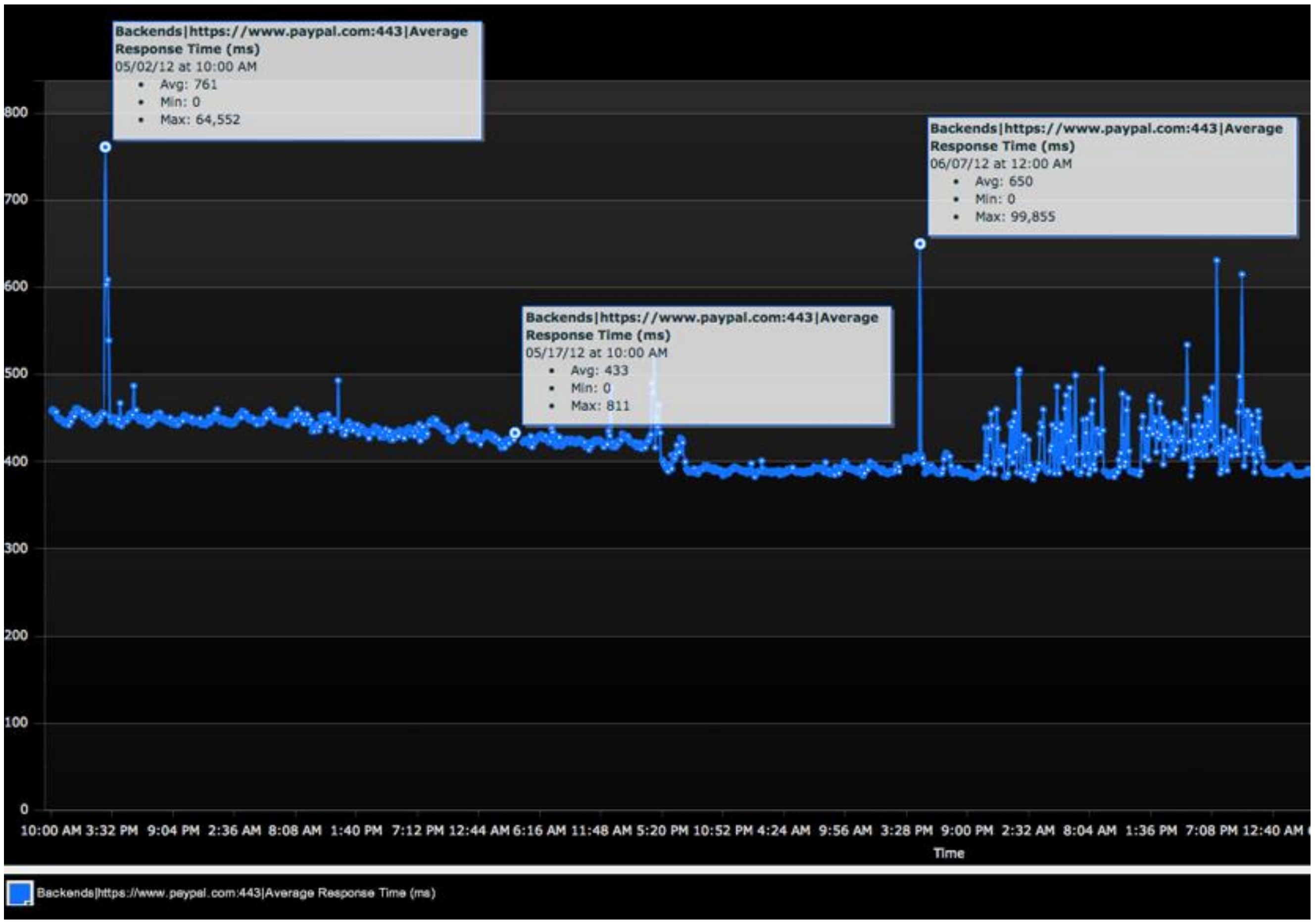


Figure (2.12) PayPal response time (Mappic, 2012)

If the application relies on one or more 3rd party web services, you should periodically check and report what level of service you are receiving each week. That way, you can truly understand your service provider QoS and its impact on your end user experience and application performance. You can also keep your service providers honest, with complete visibility of whether QoS is improving or degrading over time as service outages occur and are fixed.

2.9 Summary

In this chapter, we discussed a summary about web service in general and about RESTful service in particular and after studying quality of service and response time we used this information to specify the best time for RESTful service which we will use to find the required time to request the service for a second time, thus reaching guard time by sending multiple requests while service is still working, this will consume server's resources and cause DOS.

Chapter 3
Related Works

Chapter 3
Related Works

RESTful Services is a new and hot topic in web services field, so there is lack of papers in this field. Thus our approach of finding related works is looking for topics related to DOS attacks in general and RESTful security concerns.

3.1 Approaches to Prevent DOS Attack in Web Services

Irfan siddavatam (Siddavatam & Gadge, 2008) propose comprehensive set of test mechanism to detect attacks on web services. The study includes algorithms section to analyze SOAP requests through a number of tests and find out if it can transform into attack on web service like cipher test to find if the header element does not contain cipher or not matched with SOAP requests and reply attack test to check the time at which the last time request was received in persistent, also set guard time period to limit request in frame by admin configuration and preventing buffer overflow attack. In our approach we add dynamic different guard time for each action, thus computing time is different from one resource to another.

Zhang (2011) provide measurement to prevent DOS attack, this study analyzes the DOS attack prevention principles and gives an thorough analysis of existing prevention techniques, proposed to prevent DOS attacks by adding certification system defense that define the identity for each client, this limit the probability of attacks outside the defined clients but it is costly, in our approach we can define the identity of authorized clients as condition to generate new tokens, therefore we can apply the study of Ramin (Fouladi, 2013) that define frequency characteristics for DOS into high speed intrusion detection systems that support RESTful web services architecture as Mohsen (Rouached, 2013) propose to use RESTful Web services for coordinating heterogeneous entities of a high speed distributed IDS then detect and prevent any DOS attacks .

Cotroneo, Peluso, Romano, Ventre, and Napoli (2002) propose a new protocol that can protect against DOS attack moreover it detect the source of the attacker even there is a spoofed IP. Our approach is algorithmic; however, we can provide union solution with hardware for RESTful in future work.

3.2 Approaches to Prevent Attack in Restful Web Service

Elkurd and Barhoom (2015), proposed an algorithm to prevent buffer overflow DOS attack in RESTful services by comparing the difference between current time for RESTful request and the time for previous RESTful request with the current process computing time, then increase the token live time for request depending on the expected time for next request. Unfortunately, this solution supposes the current process computing time for RESTful request and expected time for next request as a constant. This will represent two problems, the first one: is to assume the process computing time as a constant because the difference between current request time and previous request time might take more than a constant or less than it, if the services takes less than assumed constant the approach will reject the next request which causes a noise for the client and might lose him. On the other side if the difference time takes more than a constant this will give a chance of suspicious repeated requests which lead to overload on service resources and additional cost on service provider. The second problem assumes the expected time for next RESTful request as a constant because our approach depend on this time to increase the token life time which considered as a security breach when the token time increases more than real need.

In our solution we suggest an approach to find the current process computing time using several experiments and analyze its results and estimate the expected time for next RESTful request using our equation then handle client request tokens refresh time and prevent suspicious repeated requests for each client.

Our approach will cover the enhancement for preventing slow connection attack which means that If a RESTful request is not completed, or if the transfer rate is very low, the Web server keeps its resources busy waiting for the rest of data which leads to DOS or cost more money.

Serme, Oliveira, Massiera, and Roudier (2012) provide a security protocol to make message security implementation by encrypt message exchange between the provider and client throw new encrypted headers in the http request and response in order to meet RESTful principles. While the idea is novel and provide security for RESTful services equivalent to WS-Security but it always need to overload request and response with extra headers and this is drawback.

Furthermore the study Femke (Backere et al., 2014) compare the current known security mechanism and the drawbacks for each one and how far it adaptive the RESTful (stateless) standards. Technically, the proposed security mechanism try to harmonizes the RESTful principles and the lightweight security techniques; it assume no need to encrypt each transmitted message to enhance the performance but only the sensitive information to avoid sniffing; in the login it use CA (certificate is signed by a certificate authority) and hash password H(pw) to transmit it securely and check the password with salt in the backend H(H(pw)+salt), after successful login TLS token is sent back to the client with expire date. This study is not clear of how digital certificate are created and send it to the user and used with different device. In our approach we try to adapt the REST design as possible, by save the needed information for security

Cheng, Laih, Lai, Chen, and Chen (2008) presents an on-line user-behavior surveillance system that detects the malevolent user ‘s behavior and web application attacks using Embedded Markov model (EMM). The EMM proposed in this paper is a two-phase Markov model. The first phase Markov model is used to establish the normal model for every attribute. The second phase is used to build the user–behavior surveillance model for detecting the abnormal visiting behavior.

Park, Iwai, Tanaka, and Kurokawa (2014) show analysis for Slow Read DOS attack which is one of sophisticated DOS techniques. Our approach will find a way to prevent this attack especially that it causes buffer overflow attack.

3.3 Quality of web services (QoS) to know suitable response time

(Jiang, Lee, & Hu, 2012) present one such a largescale longitudinal analysis of publicly available web services of SOAP-based and RESTful types. For the period of roughly one year and from five different world-wide locations, they closely monitor the ups and downs of various basic properties of web services and their QoS values using a total of 825,132 real web services. One of the important factors of QoS is response time of service, the study considers that the service is failed when response time is greater than 60 seconds, and the acceptable average response time is between 100 and 1,000 MS (for text message size between 0.5 K and 16 K). We have a study about response time to ensure that it will not affect the quality of service during the design of our Restful service and our approach.

3.4 Summary

In this chapter we discussed the related work about approaches to prevent DOS attack in web services specific in RESTful services and the difference between our approaches and the other, we study the quality of service to make sure that our approach not violated the factor of QoS specially the time response of our RESTful services.

The main idea in our thesis is to define the expected time for next RESTful request by using several experiments on RESTful services with different size of data to get the general equation for calculate the current computing time and get next request time according to generated equation.

Chapter 4
Proposed Approach

Chapter 4
Proposed Approach

In this chapter we present our proposed approach for estimating time for next request of RESTful service to prevent DOS attack on RESTful Services.

The main idea of our proposed approach is to get equation for estimate time of next request on RESTful service, this equation will be generated from the analysis of huge requests on RESTful service with different sizes of data.

4.2.1 Design of Proposed Approach

Our goal is to prevent DOS attack on RESTful services through finding the next request time for RESTful service and substitute our formula in the following approach. Figure 4.1 Display the developed algorithm to prevent buffer overflow DOS attack. The algorithm checked the valid token to identify the legal clients, then the request time is checked after that it saved as previous time to compare it with the new request. If the difference between the new request time and the previous time are less than the process computing time, the request is neglected.

We need to clarify some terminology in the proposed algorithm which is defined in Table 4.1

Table 4.1) Terminology in developed approach

Allowance Time	Fixed time wasted through communication and request
Process Computing Time	Current Process computing time (service time).
Next Expected Time	Next expected request maximum computing time
Guard Time	Difference time between current and previous RESTful request time

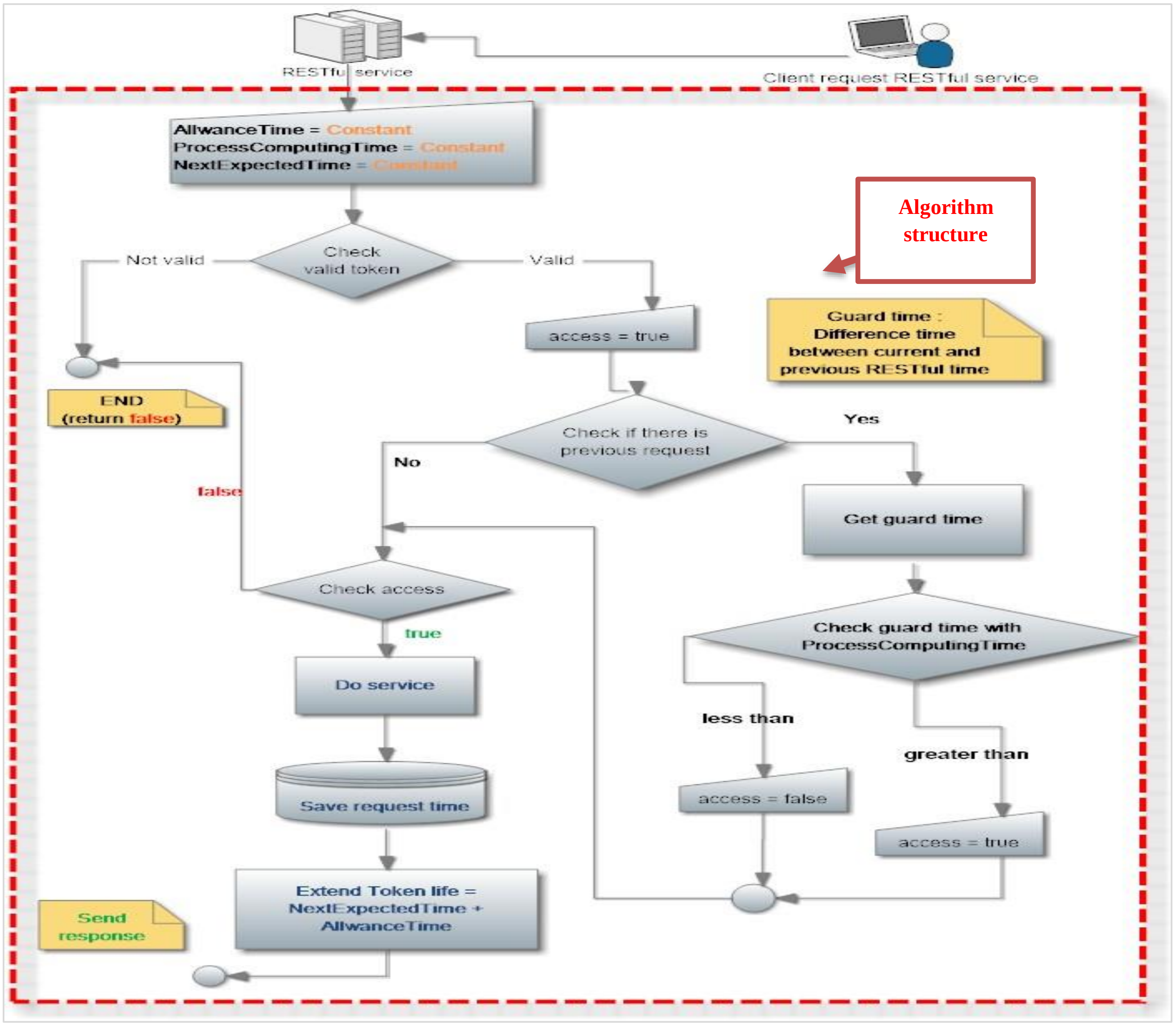

RESTful service
Client request RESTful service
AllwanceTime = Constant
ProcessComputingTime = Constant
NextExpectedTime = Constant
Algorithm structure
Check valid token
Not valid
Valid
access = true
Guard time : Difference time between current and previous RESTful time
END (return false)
Check if there is previous request
Yes
No
false
Get guard time
Check access
true
Check guard time with ProcessComputingTime
Do service
less than
greater than
Save request time
access = false
access = true
Extend Token life = NextExpectedTime + AllwanceTime
Send response

Figure (4.1) An approach of prevent DOS attack on RESTful services

The next request time in the algorithm is assumed as constant so we substitute the next request time with the following formula:

$$T_{nx} = \text{Response Time } (T_{re}) + \text{Allowance time } (W) + \text{Process Computing Time}$$

The response time is calculated according to the size of retrieved data from RESTful service negating the processing time. We work on enhancing the algorithm to be as following steps:

1. Calculate the size of retrieved data, find the response time of RESTful service and add the current time of current request to it then save it as previous time.

2. After that when the next request come, the algorithm will get the previous time and compare it with current time: if it is greater, the algorithm will reject the request. If not, the algorithm will process it and save the next request time.

Figure 4.2 shows the enhanced algorithm where we substitute the constant for next request time with our formula depending of retrieved data from RESTful service.

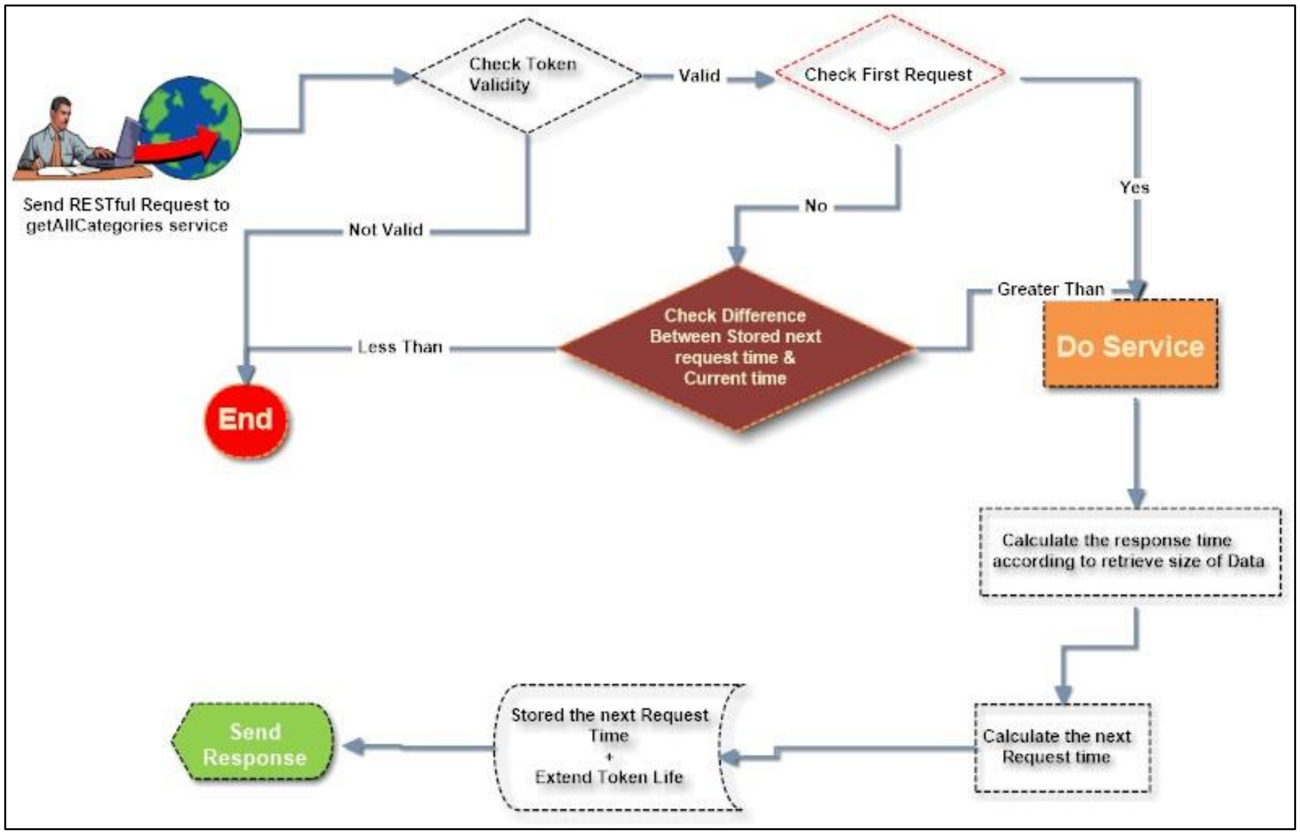

Send RESTful Request to getAllCategories service
Check Token Validity
Valid
Check First Request
Not Valid
No
Yes
Greater Than
Check Difference Between Stored next request time & Current time
Less Than
Do Service
End
Calculate the response time according to retrieve size of Data
Calculate the next Request time
Stored the next Request Time + Extend Token Life
Send Response

Figure (4.2) Enhance Algorithm with our proposed approach

Our approach consists from two following main phases:

Phase 1: Preparation which aims to prepare data and environment of our approach described in section 4.1.

Phase 2: processing the requests include the evaluation for the results and applying results on Hussam's (Elkurd & Barhoom, 2015) algorithm to prevent buffer overflow DOS, phase 2 is described in section 4.2.

However, the steps look straightforward we face many challenges to decide how the approach defined with reasonable justification depends on previous studies or a real experiment.

Phase 1
Preparation
Phase 2
Processing & Analysis

Figure (4.3) Phases of proposed method

4.1 Preparation

Our proposed model need to preparation phase. This phase consists from tow section, the first is to prepare data as JSON files, this step needs to multi-steps to get the required data we described it later in this chapter, second section of preparation phase is preparing and configure the environment, thus we perform the following steps in preparation phase:

Section 1: Data Preparation

1. Collect data: In this step we collect different data from Mashape Application described in section 2.5. The data retrieved is consists from array of objects, each object contain attribute such as name, domain, tagline, address. Figure 4.2 shows the collection data

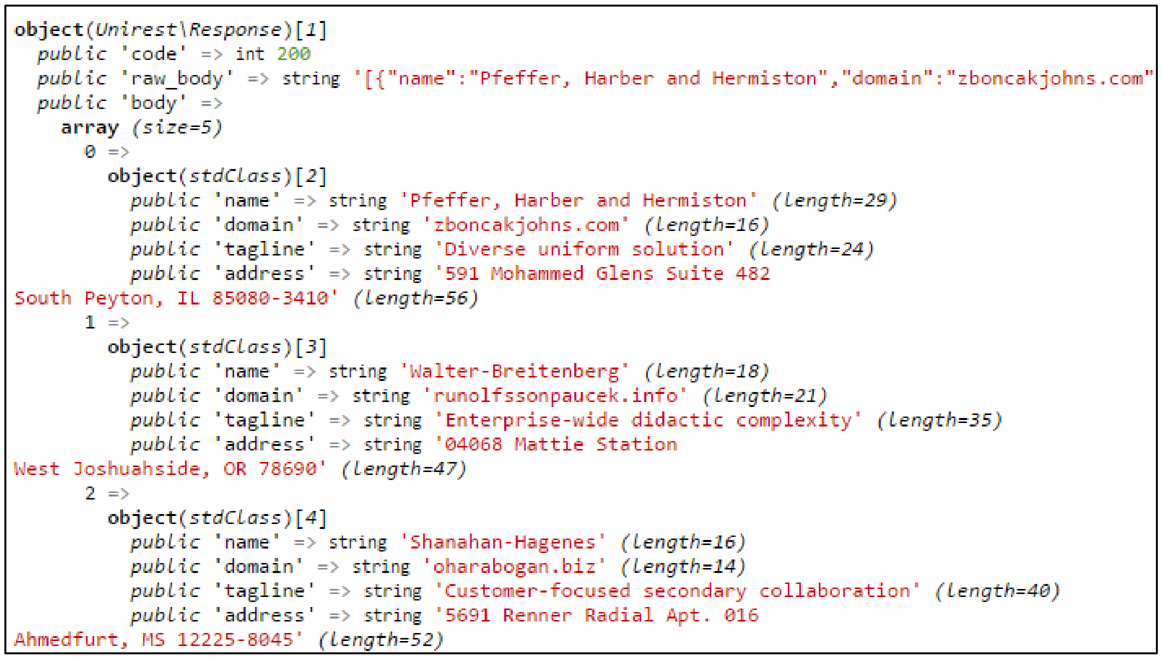

```
object(Unirest\Response)[1]
  public 'code' => int 200
  public 'raw_body' => string '[{"name":"Pfeffer, Harber and Hermiston","domain":"zboncakjohns.com"
  public 'body' =>
    array (size=5)
      0 =>
        object(stdClass)[2]
          public 'name' => string 'Pfeffer, Harber and Hermiston' (length=29)
          public 'domain' => string 'zboncakjohns.com' (length=16)
          public 'tagline' => string 'Diverse uniform solution' (length=24)
          public 'address' => string '591 Mohammed Glens Suite 482
South Peyton, IL 85080-3410' (length=56)
      1 =>
        object(stdClass)[3]
          public 'name' => string 'Walter-Breitenberg' (length=18)
          public 'domain' => string 'runolfssonpaucek.info' (length=21)
          public 'tagline' => string 'Enterprise-wide didactic complexity' (length=35)
          public 'address' => string '04068 Mattie Station
West Joshuahside, OR 78690' (length=47)
      2 =>
        object(stdClass)[4]
          public 'name' => string 'Shanahan-Hagenes' (length=16)
          public 'domain' => string 'oharabogan.biz' (length=14)
          public 'tagline' => string 'Customer-focused secondary collaboration' (length=40)
          public 'address' => string '5691 Renner Radial Apt. 016
Ahmedfurt, MS 12225-8045' (length=52)
```

Figure 4.4) Sample of collect different data from Mashape apps

2. Clean and prepare data as JSON Format.

3. Prepare cleaning JSON data in different sizes JSON file.

Section 2: Environment Preparation

1. Design RESTful service to perform testing.

2. Select environment testing over cloud.

3. Create Client Script to send requests.

After perform the 2 sections the data and environment are ready to process requests and evaluate the results to prevent buffer overflow DOS. However, we may continuously have made modifications on the data set such as size in order to enhance the results, below figure 4.5 shows conceptual diagram for preparation phase. In the next para graphs we discuss each step in details.

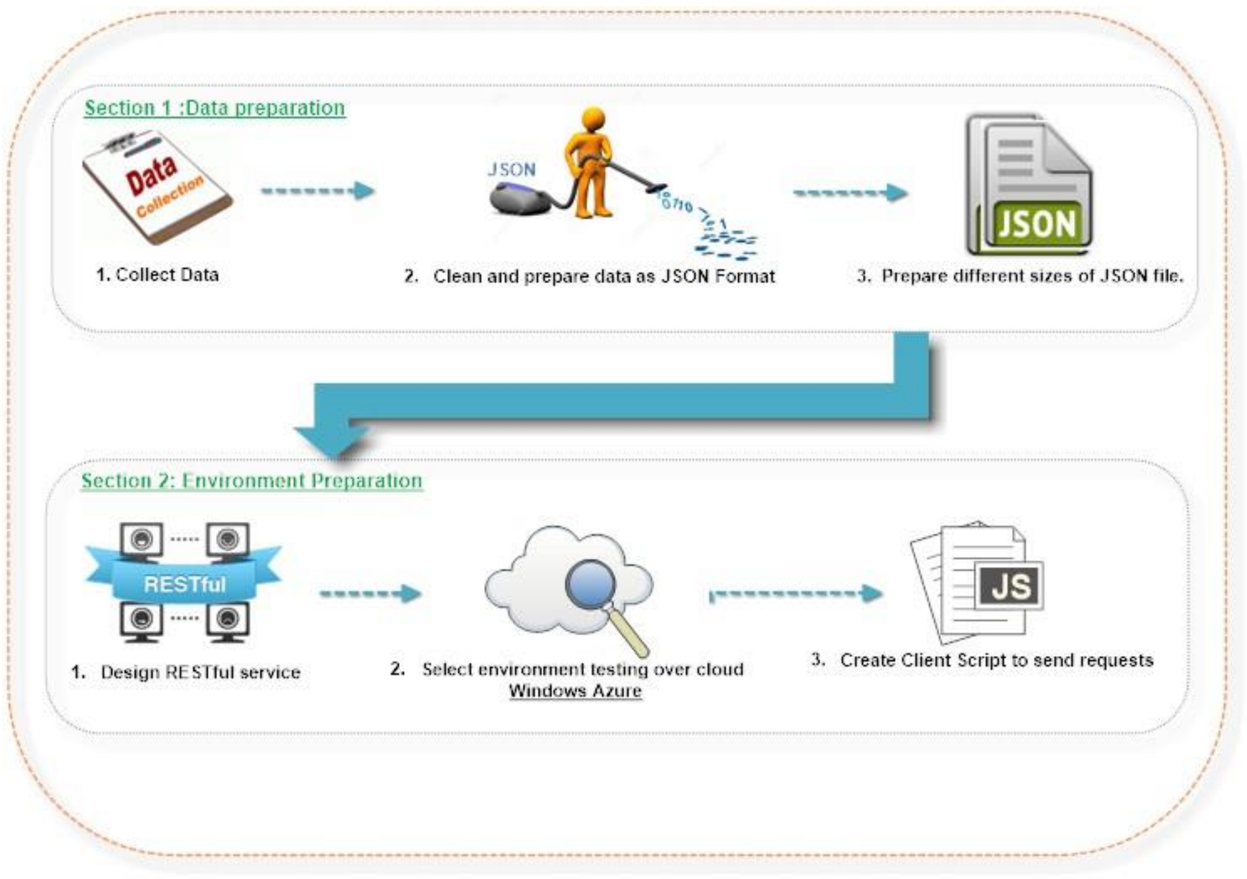

Section 1 :Data preparation
Data Collection
1. Collect Data
JSON
2. Clean and prepare data as JSON Format
JSON
3. Prepare different sizes of JSON file.
Section 2: Environment Preparation
RESTful
1. Design RESTful service
2. Select environment testing over cloud Windows Azure
JS
3. Create Client Script to send requests

Figure (4.5) Preparing Data and environment

4.1.1 Collect and prepare Data

Our approach need to use JSON file in different sizes. The first part is collect data, thus we have developed a PHP script to collect data throw dummy data API'S, DumDataCol collect data and save it in files, therefore the files are assembled incrementally in a single data warehouse were the data are prepared and cleaned to be ready to use it in our services, we have discussed DumDataCol in Section 5.1

The second part is to clean and formatting data as JSON. DataJsonTra scripts take the dummy data and clean it from any unused and blank data then transfer it to JSON formatting, after that generateDiffSize script is responsible to generate different size of JSON files.

In this steps we generate different size of JSON file with different data (from 16 K size to 5 M). Later in chapter 5, every step will be discussed in details.

4.1.2 Environment preparation

First step in environment preparation is to design restful services, that our proposed approach depends on it, then prepare cloud environment to test our approach. In the next para graphs we discuss each step in details.

4.1.2.1 Design RESTful services

We will design two types of RESTful services (GET and POST) called getAllCategories and postCatData, our services are declined process time and implement using PHP Object Oriented to be suitable with Azure platform. Chapter 5. Describe this in details.

4.1.2.2 Select environment over cloud

Our proposed approach is designed to work on a cloud environment. For this purpose, we have two choices, either use a real test beds such as Windows Azure or use simulation tools to simulate a cloud environment. In our work we use real environment windows Azure. The reason of our choice is that Azure is very friendly and easy to use with their portal and web Apps that it's an easy-to-manage, scalable, highly secure, and highly available solution for running presence websites. We will set up our services as a Website. Chapter 5 will describe how we deploy our services using Azure in details.

4.1.2.3 SendClientReq script

We developed client script to send REST asynchronous request for our RESTful services called SendClientReq. The goal of the script is to send several requests to record the response time for every request, then analyze a set of requests to get acceptable response time. We applied this script for different size of data. Table 4.1 shows the pseudo code for SendClientReq script.

Table (4.2) Pseudo code for SendClientReq

```
SendClientReq 4.1
Output:   returnData JSON data
1.  SET n to number of repeated requests
2.  FOR counter = 1 to n-1 do
3. CALL RESTfulService()
4. RETURN returnData            //return data from RESTful service formatted as JSON
```

4.2 Processing Requests and Results

The next stage after preparing the data and environment is processing results and enhancing the results. The phase consists of 5 steps as follow:

1. Send requests to RESTful service and save results in HAR files.
2. Process HAR files to get data in Excels
3. Evaluate consuming time for defined data sizes
4. Analyze results and justification
5. Evaluate results Using Hussam's Algorithm to prevent DOS attack

The conceptual diagram in Figure 4.7 below shows the main steps in classification phase.

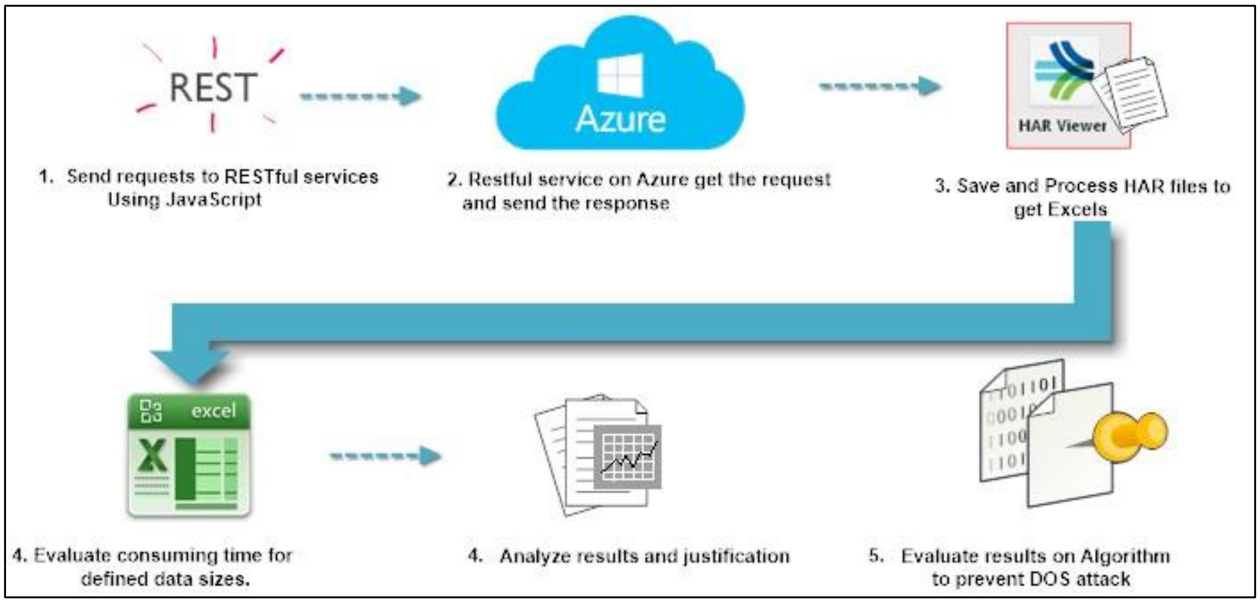
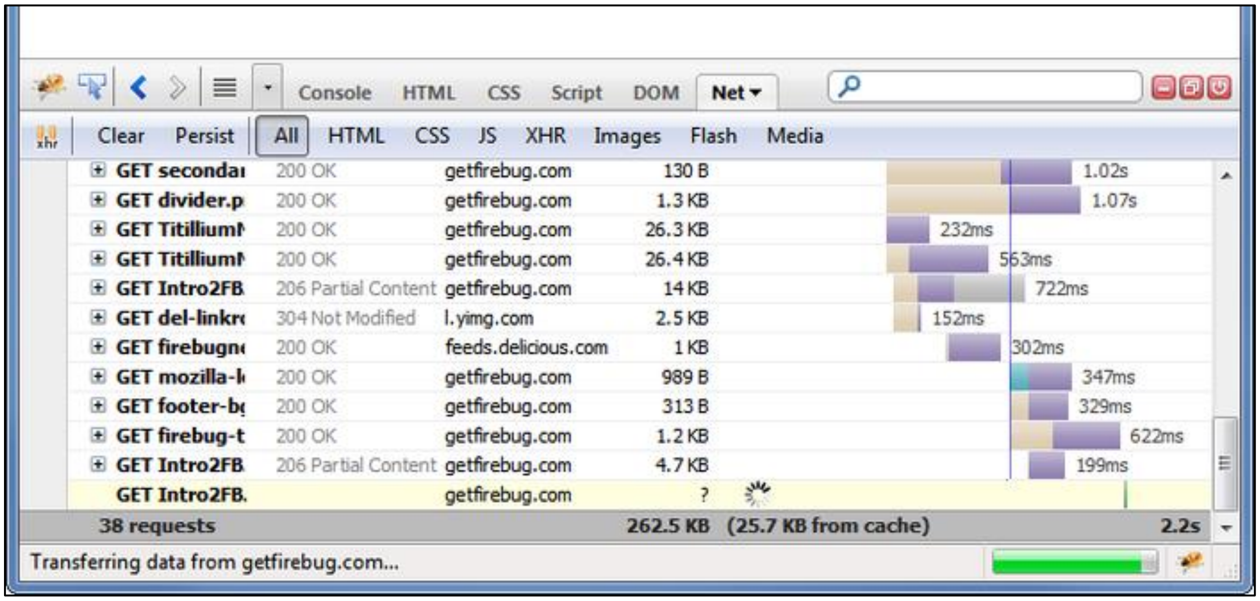

REST
Azure
HAR Viewer
1. Send requests to RESTful services Using JavaScript
2. Restful service on Azure get the request and send the response
3. Save and Process HAR files to get Excels
excel
4. Evaluate consuming time for defined data sizes.
4. Analyze results and justification
5. Evaluate results on Algorithm to prevent DOS attack

Figure (4.6) Process and evaluate results

4.2.2 Send requests to RESTful services

We will apply SendClientReq scripts to send requests from client to RESTful services. In this step we use different data sizes (16 K, 32 K, 64 K, 128 K, 250 K, 512 K, 1 M, 2 M, 3 M, 4 M, 5 M) and send 50 requests for each size of data. The request must have some attributes like the type of request is GET Request with no cache and asynchronous, the details of each requests are saved in HAR file. Using Mozilla FireBug tools. Firebug integrates with Firefox to put a wealth of development tools at your fingertips while we browse. We can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page. We use NET window in firebug to monitor time response for each requests and save it. Figure 4.8 shows the NET window in FireBug tools.



GET secondai	200 OK	getfirebug.com	130 B	1.02s
GET divider.p	200 OK	getfirebug.com	1.3 KB	1.07s
GET TitilliumM	200 OK	getfirebug.com	26.3 KB	232ms
GET TitilliumM	200 OK	getfirebug.com	26.4 KB	563ms
GET Intro2FB	206 Partial Content	getfirebug.com	14 KB	722ms
GET del-linkr	304 Not Modified	l.yimg.com	2.5 KB	152ms
GET firebugn	200 OK	feeds.delicious.com	1 KB	302ms
GET mozilla-l	200 OK	getfirebug.com	989 B	347ms
GET footer-b	200 OK	getfirebug.com	313 B	329ms
GET firebug-t	200 OK	getfirebug.com	1.2 KB	622ms
GET Intro2FB	206 Partial Content	getfirebug.com	4.7 KB	199ms
GET Intro2FB.		getfirebug.com	?	
38 requests			262.5 KB (25.7 KB from cache)	2.2s

Transferring data from getfirebug.com...

Figure (4.7) NET window in FireBug tool

4.2.3 Process HAR files to get Excels

When we save the NET window as discussed in the previous section, the output files are saved in HAR format. HAR (HTTP Archive) (Google Apps Tool, 2014) is a file format used by several HTTP sessions tools to export the captured data. The format is basically a JSON object with a particular field distribution. We use har2csv tools to convert our HAR files to csv formats then excel to deal with results, we change the code of this files to get our need from the HAR files like data size, waiting time, consume time.

4.2.4 Evaluate consuming time for different size

We analyze excel files to get consume time for requests for each size of data, using average and median methods, we depend on median method to get consume time because the median is the number in the middle of a set of numbers, so we can discard all abnormal records.

4.3 Summary

This chapter presented the approach for preventing DOS attack on RESTful service, therefore it shapes the approach for estimating next request time for RESTful services. The chapter described the preparation phase include the preparation of data which output several JSON files with several size of data and preparation of environment which include the design of restful services and prepare client script which is responsible for sending several RESTful requests for services. Further it takes over the processing phase, including sending requests to RESTful service and save results in HAR files. From the HAR files and used Excel methods we got the equation to find the next RESTful request time

Tnx = Response Time (Tre) + Allowance time (W) + Process Computing Time

After that we redesign Hussam’s algorithm and enhancing it according to the new formula then evaluate the equation by repeating the experiments using enhanced algorithm. The next chapter presents the experiments based on the approach.

Chapter 5
Implementation

Chapter 5
Implementation

In this chapter we discuss the implementation of the proposed approach and describe the details of the practical application.

5.1 Collect and prepare Data

Collect and prepare data consists of 3 main stages: first stage is to collect dummy different data from Mashape application and save it to file. DumDataCol script responsible to achieve this stage, the second stage is to clean and format data using DataJsonTra script, and the last stage GenerateDiffSize script responsible for generate different size according to our needs in experiment. Figure 5.1 shows the stages for collecting and preparing data.

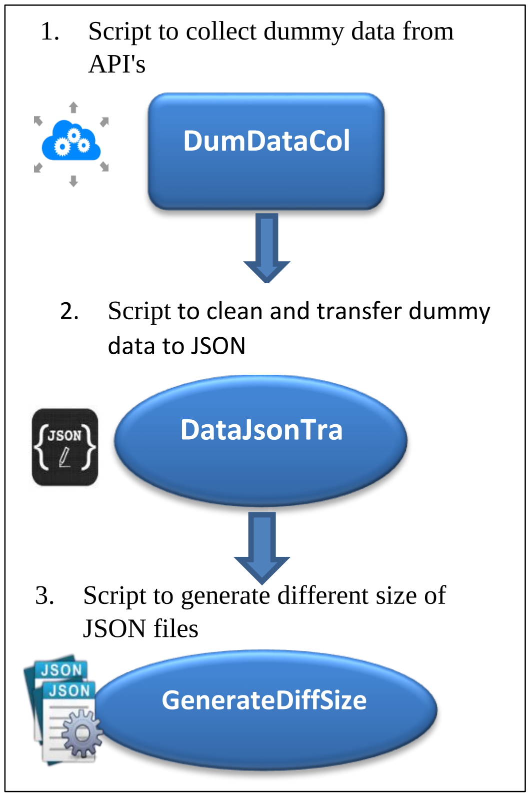

1. Script to collect dummy data from API's
DumDataCol
2. Script to clean and transfer dummy data to JSON
JSON
DataJsonTra
3. Script to generate different size of JSON files
JSON
JSON
GenerateDiffSize

Figure (5.1) Collect and prepare Data

5.1.1 DumDataCol implementation

We use DumDataCol to collect different data, DumDataCol is a web application written in PHP language that connect to dummy data API'S to get different data and save it in file as shown in Figure 5.2.

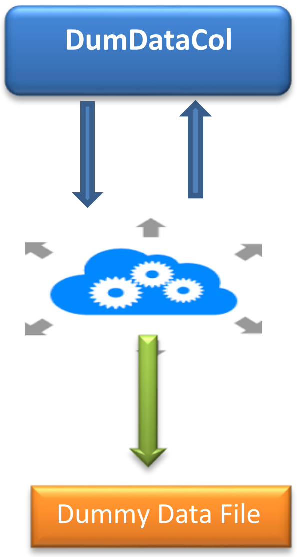


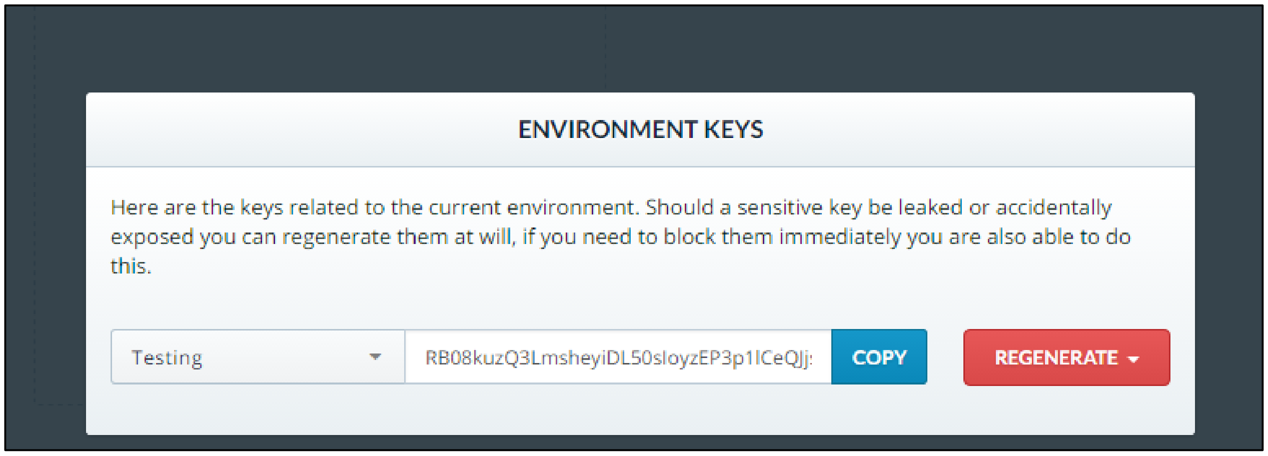
Figure (5.2) Design of script to collect dummy data from API's

First of all, we create Mashape Application on Mashape development platform in order to enable DumDataCol access Mashape through application key as we describe in chapter 2 After authenticate the app, we look to find dummy data that used as returned data from our RESTful services.

To create application, we do the following steps:

1. From the site navigation toolbar, we choose create application then named it " *dummyDataApps*"

2. From dashboard we click on our application and take the key to use it in our script.


ENVIRONMENT KEYS

Here are the keys related to the current environment. Should a sensitive key be leaked or accidentally exposed you can regenerate them at will, if you need to block them immediately you are also able to do this.

Figure (5.3) Mashape Application Key

DumDataCol scripts consist of two stages, first stage is to get data from dummyDataApps application since this application give just 5000 record of data and the second stage is to save data in file until get the required size.

Table (5.1) Shows the pseudo code of DumDataCol

```
Script 5-1  DumDataCol
Input: dataSize Size of required data
Output: dummyDataFile file of dummy data

open "dummyDataFile " for output
while size of  dummyDataFile not equal dataSize
        request  dummyDataApps
        set  dummyDataResponse from the request;
        if  dummyDataResponse contain data then
        for dataCounter:= 0 to  dummyDataResponse size do
        format dummyDataResponse[dataCounter];
        save the current  dummyDataResponse and add it to  dummyDataFile ;
    end;
end;
```

5.1.2 DataJsonTra implantation

We develop DataJsonTra script in PHP languages which is responsible to clean dummy data from unrequired data like empty variable and spatial character then transfer data to JSON and save it Figure5.4 show the input and output for scripts and Table 5.2 shows the pseudo code for DataJsonTra script.

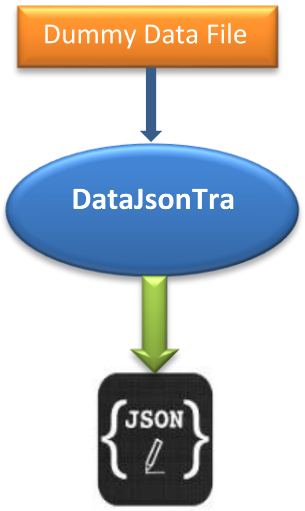


Figure (5.4) Design of script to clean and transfer dummy data to JSON

Table (5.2) Shows the pseudo code for DataJsonTra script**.**

```
Script 5-2  DataJsonTra
Input: dataFile file of data
Output: jsonFile file of JSON data

open " dataFile " for input (reading) ;
open " jsonFile " for output (writing) ;
while not end of file " dataFile "
            read data items from " dataFile ";
            set dummyData from " dataFile ";
                if  dummyData equal empty then
            delete current dummyData from " dataFile ";
```

```
        else
    format  current dummyData to JSON ;
    save the current  dummyData and add it to  jsonFile;
end;
```

5.1.3 GenerateDiffSize implementation

GenerateDiffSize scripts is responsible to create different sizes of JSON files, we need to perform our experiments on (16 K , 32 K , 64 K, 128 K , 250 K, 512 K, 1 M , 2 M , 3 M , 4 M , 5 M) sizes Table 5.3 shows the pseudo code for GenerateDiffSize script and Figure 5.5 shows the input and output for scripts.

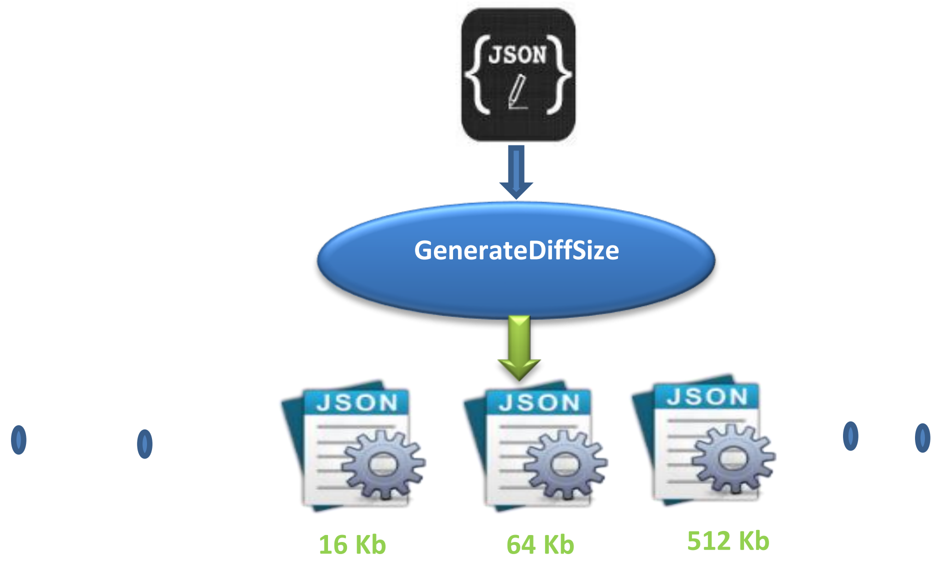


Figure (5.5) Design of script to generate different size of JSON files

Table (5.3) Shows the pseudo code for GenerateDiffSize script

```
Script 5-3  GenerateDiffSize
Input: dataFormatedFile file of data
Output: jsonFileDiffSize different sizes of JSON files
    set requiredSizeArr array of sizes
    open " dataFormatedFile " for input (reading) ;
```

```
for  indexCounter:= 0 to requiredSizeArr size do
open " jsonFile " for output (writing) ;
        while not end of file " dataFormatedFile "
read data items from " dataFormatedFile ";
save the current  data items and add it to  jsonFile;
        if size of jsonFile equal  requiredSizeArr [indexCounter]
        break;
        end;
end;
```

5.2 Prepare environment

5.2.1 RESTful services implementation

The following four steps describe our services implementation :

Step 1: Initialization

The first part of design our services is basic initialization. This part of the script defines the service's settings, such as whether a secure HTTPS connection is required and whether the services is username and password protected. Additionally, the API initialization will define the API response codes. Finally, the initialization contains a function definition for the "deliver_response" function. This function controls the HTTP response codes, sets the Content-Type header and formats the data as JSON. The "deliver_response" function is described in more detail in Step 3.

Step 2: Process Request

The second step of the web service script is the meat of the API. This is where the requested data is gathered by the application. In our work, the response data is JSON data files which is prepared in previous section 4.2.1, the service take parameter to determine the size of JSON file to retrieve it.

Step 3: Deliver Response

The final action of our web service script is to deliver a response in JSON format. The "deliver_response" function first defines several basic HTTP response codes. After the response codes are defined, The JSON response is complete according to size of JSON file.

RESTful principles provide strategies to handle CRUD (Heller, 2007) actions using HTTP methods mapped as follows:

- GET /tickets - Retrieves a list of tickets
- POST /tickets - Creates a new ticket

We need to set up our service environment which is composed of 2 files, an application script (index.php) and a URL redirection file (.htaccess). The URL redirector will allow users to connect to the service using pretty URLs. The .htaccess file requires the mod_rewrite module on an Apache server. The API can still work if mod_rewrite is not available, but URLS will need to be formatted like:

http://ourdomainOnAzure/Controller/serviceName/Size

Table (5.4) Shows the .haccess code

```
RewriteEngine On
    #Route to Main file
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{URL} !-d
    RewriteRule ^([^/]+)/([^/]+)?/(.*)$ testApp/webroot/index.php?ttag=$1&rt=$2/$3 [L]
```

We will design 2 type of RESTful services (GET and POST), our services are declined process time and implement Using PHP Object Oriented to be suitable with Azure platform.

Table 5.5 present the pseudo code for GET RESTful service called getAllCategories. This service takes a parameter to determine the size of returned data, then take the data of JSON file and returned it to client.

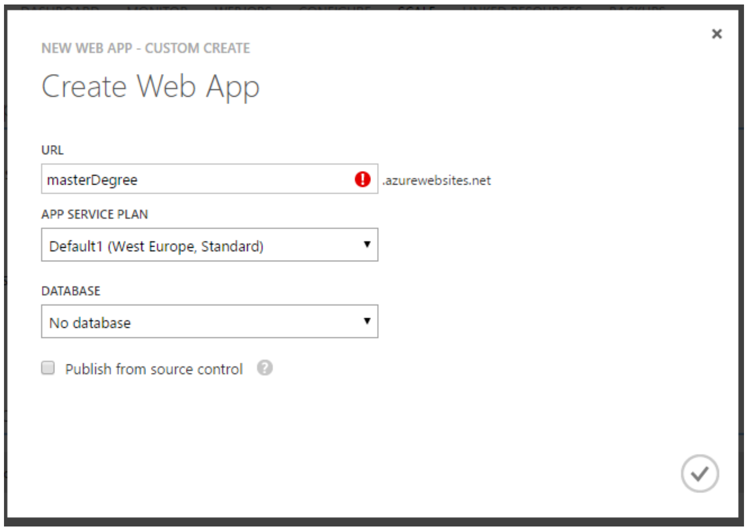
Table (5.5) Pseudo code for GET RESTful service

```
Service 5-5 getAllCategories
Input: dataSize Size of returned
Output: returnData JSON Dummy data
        define HTTP request
        set HTTP response
        set HTTP Response Content Type to JSON
        if dataSize greater than 0
            open " Jsonfile " for input (reading) ; // according to size
            read data items from " Jsonfile " and set to returnData;
         end;   // Error in service
        return returnData     //return data in file formatted as Json
```

5.2.2 Windows Azure environment

We need to prepare our environment on windows Azure by perform the following steps:

1- Create web app as shown in figure 5.6 we choose the nearest datacenter which is located in West Europe


NEW WEB APP - CUSTOM CREATE

Create Web App

URL: masterDegree .azurewebsites.net

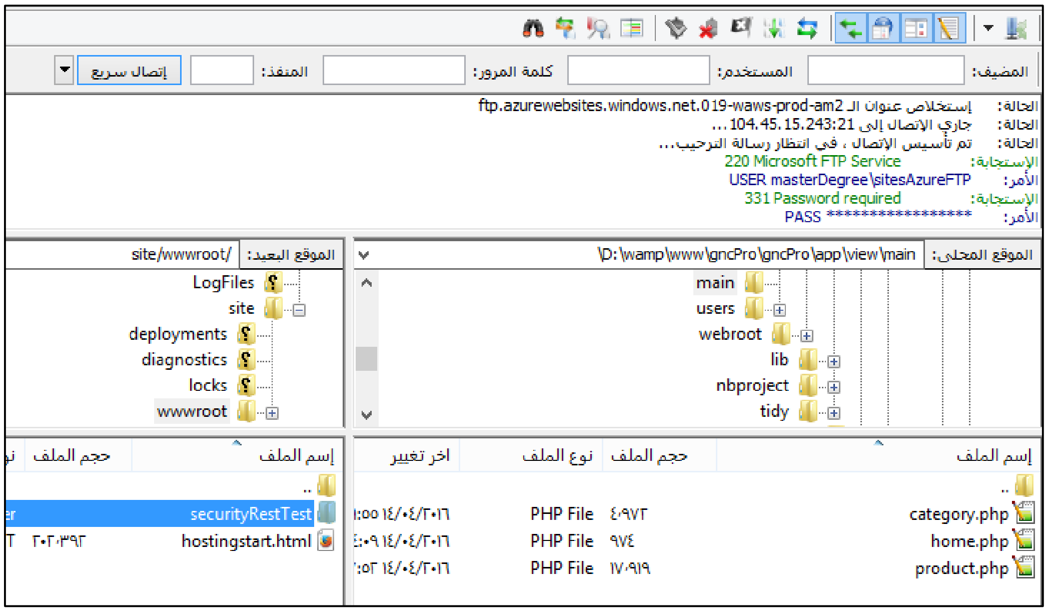
APP SERVICE PLAN: Default1 (West Europe, Standard)

DATABASE: No database

Publish from source control

Figure (5.6) Create web app on windows azure

2- Upload our RESTful services on web app by using the ftp program such as FileZilla and transfer all files using it. Figure 5.7 show the transfer files into Azure.


المضيف: المستخدم: كلمة المرور: المنفذ: إتصال سريع

```
الحالة:      إستخلاص عنوان الـ ftp.azurewebsites.windows.net.019-waws-prod-am2
الحالة:      جاري الإتصال إلى 104.45.15.243:21 ...
الحالة:      تم تأسيس الإتصال ، في انتظار رسالة الترحيب...
الإستجابة:   220 Microsoft FTP Service
الأمر:       USER masterDegree\sitesAzureFTP
الإستجابة:   331 Password required
الأمر:       PASS ******************
```

الموقع البعيد: /site/wwwroot

```
LogFiles
site
  deployments
  diagnostics
  locks
  wwwroot
```

الموقع المحلى: D:\wamp\www\gncPro\gncPro\app\view\main\

```
main
users
webroot
lib
nbproject
tidy
```

إسم الملف	حجم الملف
..	
securityRestTest	
hostingstart.html	٢٠٢٠٣٩٢

إسم الملف	حجم الملف	نوع الملف	اخر تغيير
..			
category.php	٤٠٩٧٢	PHP File	١٤/٠٤/٢٠١٦ :٥٥
home.php	٩٧٤	PHP File	١٤/٠٤/٢٠١٦ :٠٩
product.php	١٧٠٩١٩	PHP File	١٤/٠٤/٢٠١٦ :٥٢

Figure (5.7) Transfer RESTful service into web app on Azure

5.3 Summary

This chapter presented the implementation of our approach, it takes over the phases of the approach described in Chapter 4, from the preparation phase to the processing. In preparation phase we implements three algorithms the first one to collect data called “DumDataCol “, second called “DataJsonTra” which is responsible to clean dummy data from unrequired data, and the third algorithm called “GenerateDiffSize” to create different sizes of JSON files which used in our RESTful services. In addition, the chapter presented the implementation of the RESTful services and show how we setup our work on windows Azure. In the next chapter we introduce the experiment and results.

Chapter 6
Experiments and Results

Chapter 6
Experiments and Results

In this chapter we will present the experiments on the proposed approach. The experiments are divided into two parts. The first part presents a study of response time for RESTful requests using different size of retrieve data. The second part contains the experiments which test and evaluate the results and equations of the proposed method.

For every experiment, firstly we will present in details the objective, and the result for every experiment. And then at the end of each part, we summarize and discuss result of the experiments. All experiments are done on windows Azure environment.

6.1 Experiment Part I

In this part the main aim of experiment is studying the response time of RESTful requests for different size of Data begin from 16K end with 5 M to generate equation to get the response time so we can indicate for next request time and substitution the formula in Hussam's algorithm, thus we performed the experiment on windows Azure.

This experiment is done to get average response time of request for RESTful services on windows Azure. Number of request is 50 requests for RESTful service (getAllCategories) with different size of response data, the choice the 50 request because we have limit choices with our account on windows Azure, thus every request we charging money more

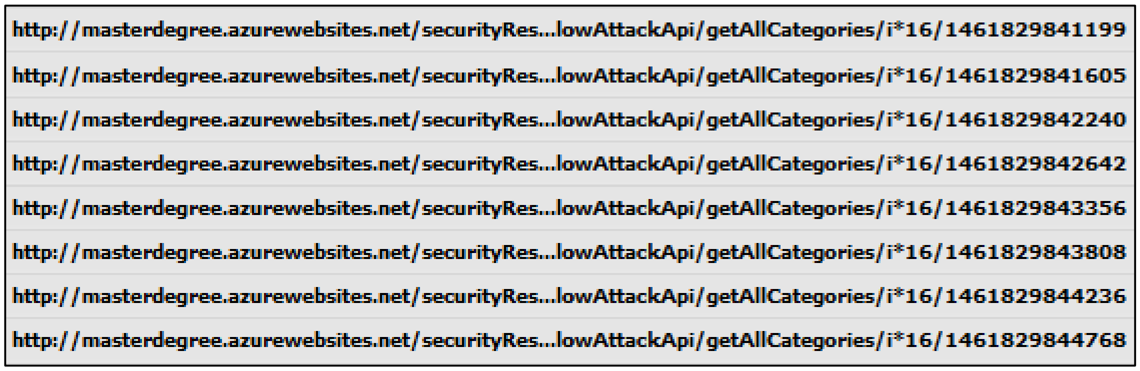
6.1.1 Experiment for 16 kb data size

In this experiment we send the request with parameter size equal 16 so the service catch the parameter and response with 16 kb of JSON data.

The link is:

http://masterdegree.Azurewebsites.net/bufferOverFlowAttackApi/getAllCategories/size*16

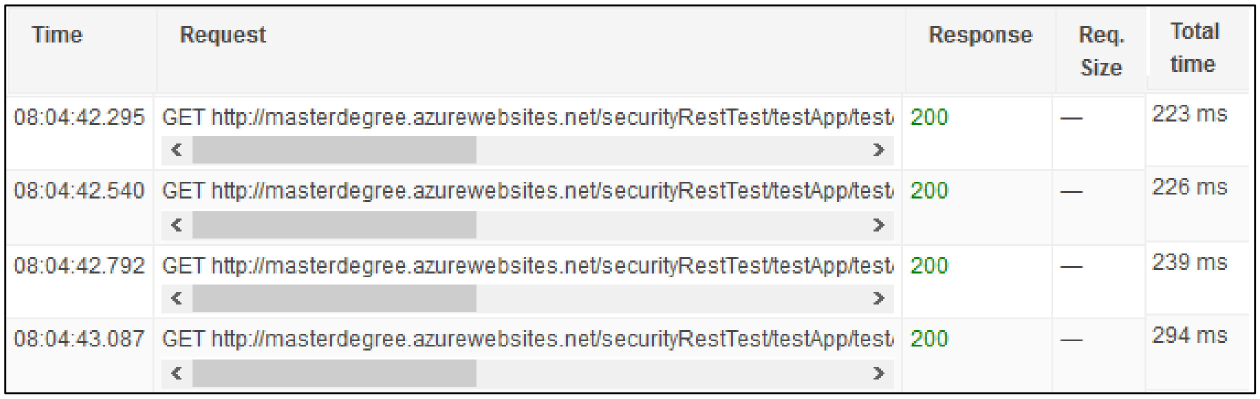
Figure 6.1 shows the requests of service on windows Azure. We capture this figure using FireBug and the request activities using windows Azure portal.



```
http://masterdegree.azurewebsites.net/securityRes...lowAttackApi/getAllCategories/i*16/1461829841199
http://masterdegree.azurewebsites.net/securityRes...lowAttackApi/getAllCategories/i*16/1461829841605
http://masterdegree.azurewebsites.net/securityRes...lowAttackApi/getAllCategories/i*16/1461829842240
http://masterdegree.azurewebsites.net/securityRes...lowAttackApi/getAllCategories/i*16/1461829842642
http://masterdegree.azurewebsites.net/securityRes...lowAttackApi/getAllCategories/i*16/1461829843356
http://masterdegree.azurewebsites.net/securityRes...lowAttackApi/getAllCategories/i*16/1461829843808
http://masterdegree.azurewebsites.net/securityRes...lowAttackApi/getAllCategories/i*16/1461829844236
http://masterdegree.azurewebsites.net/securityRes...lowAttackApi/getAllCategories/i*16/1461829844768
```

Figure (6.1) Client Request for Azure Service of 8 kb data size

Figure 6.2 shows the snapshot of HAR file recorded for this experiment that record the URL link and the time response for every requests and the all data captured from request.



Time	Request	Response	Req. Size	Total time
08:04:42.295	GET http://masterdegree.azurewebsites.net/securityRestTest/testApp/test	200	—	223 ms
08:04:42.540	GET http://masterdegree.azurewebsites.net/securityRestTest/testApp/test	200	—	226 ms
08:04:42.792	GET http://masterdegree.azurewebsites.net/securityRestTest/testApp/test	200	—	239 ms
08:04:43.087	GET http://masterdegree.azurewebsites.net/securityRestTest/testApp/test	200	—	294 ms

Figure (6.2) HAR file for RESTful request with 16 kb data size

☒ Results

We will convert the following HAR file into Excel file to get the Table 6.1 which contain all details of request, thus we note that the response time is the response from the service measured in millisecond and the waiting time is from the time of request to the time first byte is received, which involves a round trip time. There can be latency if your server away from your machine. Usually it requires 3 round trips. 1 for DNS lookup and 1 for establishing TCP Connection, 1 for request and response pair, total receive time is the response time plus the waiting time.

Table (6.1) Shows the result of RESTful requests with 16 kb

Total Receive time (ms)	**Time – Response (ms)**	**Time – Wait (ms)**
249	10	**239**
234	10	**224**
223	10	**213**
226	11	**215**
239	11	**228**
294	10	**284**
255	10	**245**
338	12	**326**
320	11	**309**
238	10	**228**
310	11	**299**
252	10	**242**

We use the average method to get the average of total response time in millisecond since we note that the average response time is 10.4 ms and the average of total receive time is 258.7 ms .

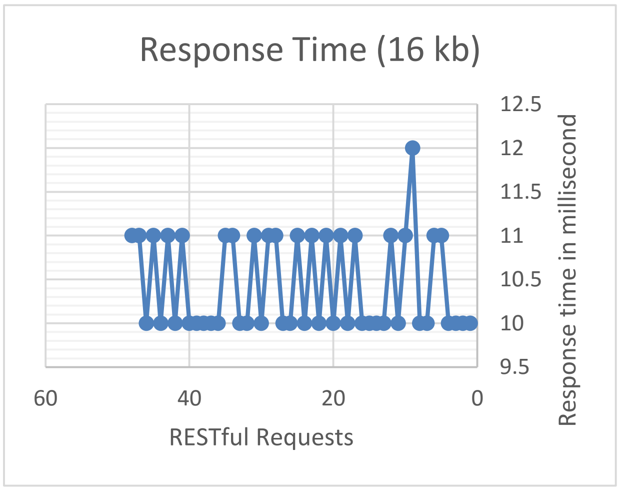


Figure (6.4) Response time of RESTful requests for 16 Kb

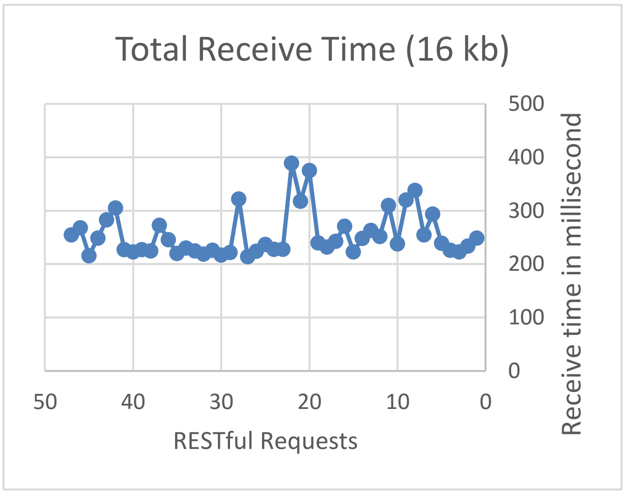


Figure (6.3) Receive time of RESTful requests for 16 kb

6.1.2 Experiment for 32 kb data size

In this experiment we send the request with parameter size*32 so the service get the parameter and response with JSON data the size of it is 32 kb.

The link is:

http://masterdegree.Azurewebsites.net/bufferOverFlowAttackApi/getAllCategories/size*32

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

☒ **Results**

After analyze the excel file that get it from HAR file, we use the average method to get the average of total response time in millisecond since we note that the average response time is 20 ms and the average of total receive time is 268.1 ms.

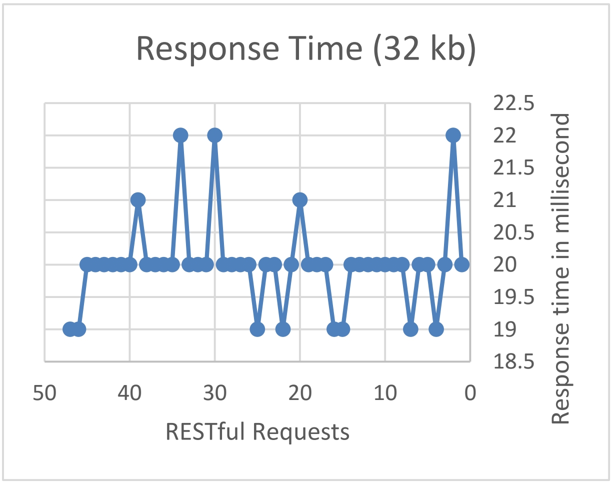


Figure (6.6) Response time of RESTful requests for 32 kb

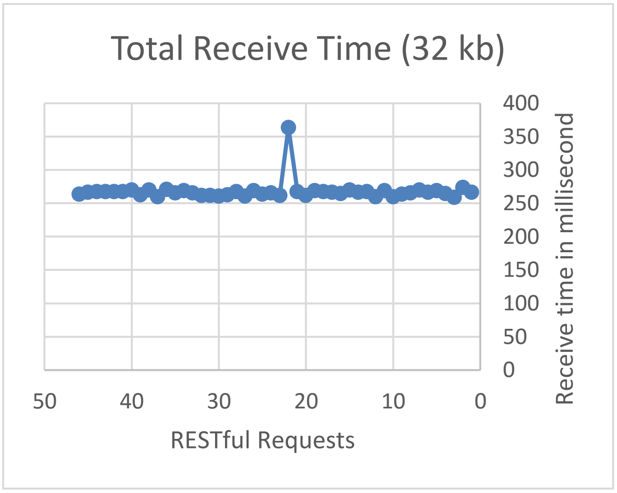


Figure (6.5) Receive time of RESTful requests for 32 kb

6.1.3 Experiment for 64 kb data size

In this experiment we send the request with parameter size*64 so the service get the parameter and response with JSON data the size of it is 64 kb.

The link is:

http://masterdegree.Azurewebsites.net/bufferOverFlowAttackApi/getAllCategories/size*64

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

☒ **Results**

After analyze the excel file that get it from HAR file, we use the average method to get the average of total response time in millisecond since we note that the average response time is 39.91 ms and the average of total receive time is 288.1 ms.

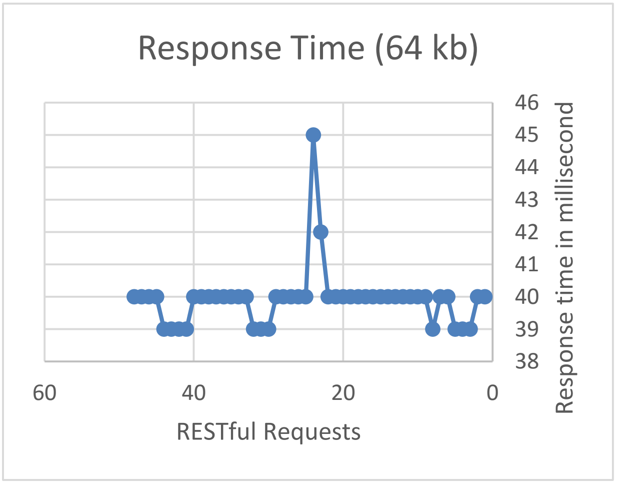


Figure (6.8) Response time of RESTful requests for 64 kb

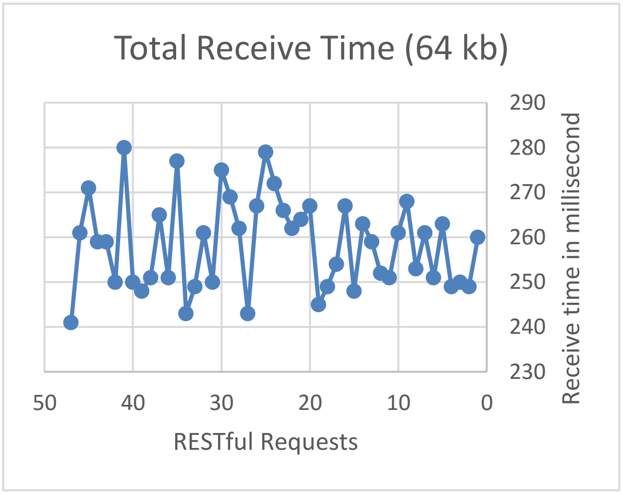


Figure (6.7) Receive time of RESTful requests for 64 kb

6.1.4 Experiment for 128 kb data size

In this experiment we send the request with parameter size*128 so the service get the parameter and response with JSON data the size of it is 128 kb.

The link is:

http://masterdegree.Azurewebsites.net/bufferOverFlowAttackApi/getAllCategories/size*128/

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

☒ **Results**

After analyze the excel file that get it from HAR file, we use the average method to get the average of total response time in millisecond since we note that the average response time is 80.09 ms and the average of total receive time is 328.6 ms .

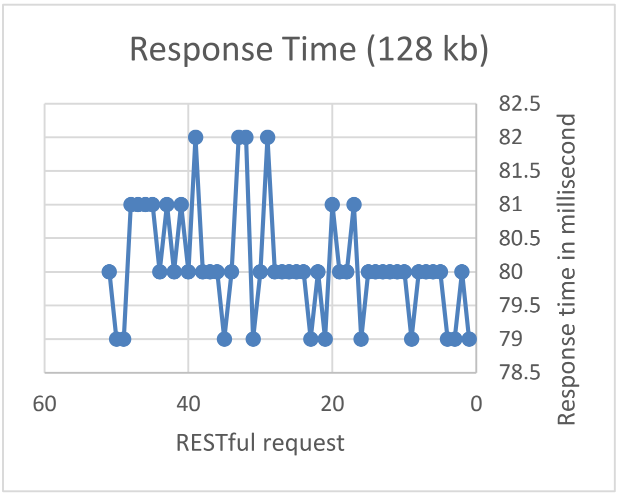


Figure (6.10) Response time of RESTful requests for 128 kb

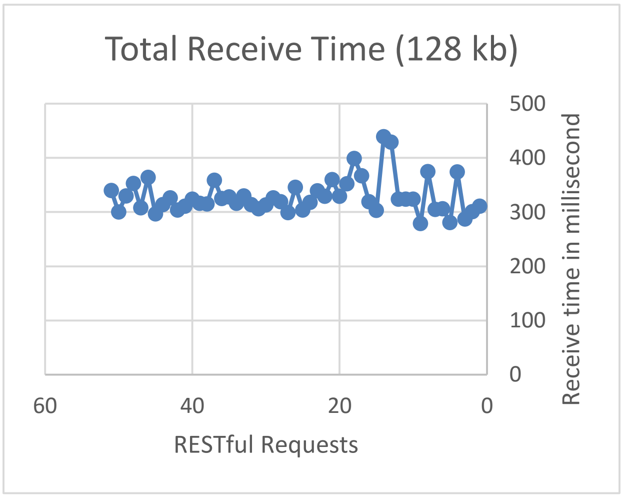


Figure (6.9) Receive time of RESTful requests for 128 kb

6.1.5 Experiment for 256 kb data size

In this experiment we send the request with parameter size*256 so the service get the parameter and response with JSON data the size of it is 256 kb.

The link is:

http://masterdegree.Azurewebsites.net/bufferOverFlowAttackApi/getAllCategories/size*256/

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

☒ **Results**

After analyze the excel file that get it from HAR file, we use the average method to get the average of total response time in millisecond since we note that the average response time is 160.07 ms and the average of total receive time is 408.1 ms .

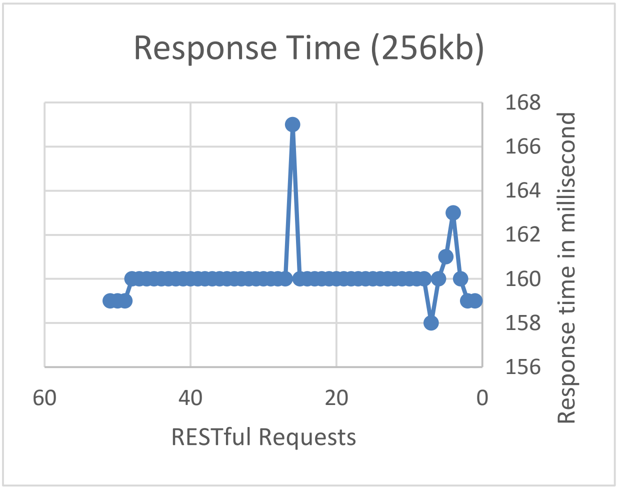


Figure (6.12) Response time of RESTful requests for 256 kb

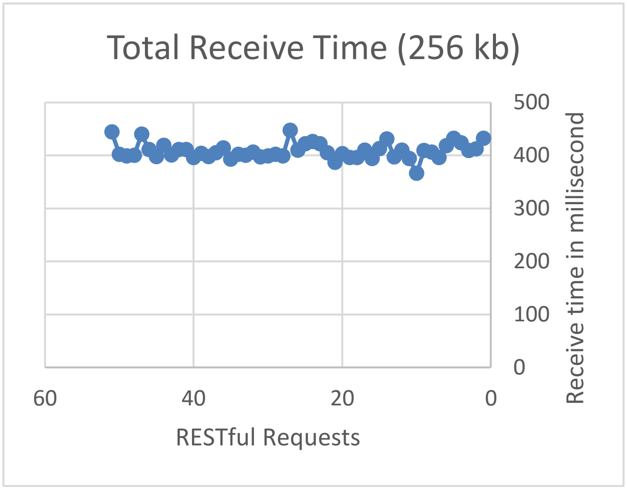


Figure (6.11) Receive time of RESTful requests for 256 kb

6.1.6 Experiment for 512 kb data size

In this experiment we send the request with parameter size*512 so the service get the parameter and response with JSON data the size of it is 512 kb.

The link is:

http://masterdegree.Azurewebsites.net/ bufferOverFlowAttackApi/getAllCategories/size*512/

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

☒ **Results**

After analyze the excel file that get it from HAR file ,we use the average method to get the average of total response time in millisecond since we note that the average response time is 319.1 ms and the average of total receive time is 566.3 ms .

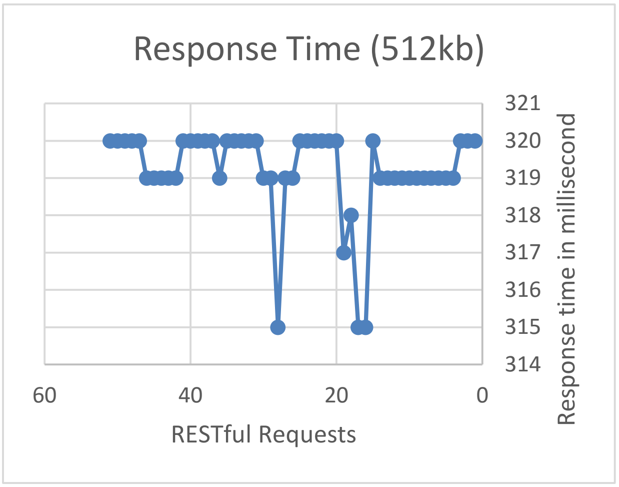


Figure (6.14) Response time of RESTful requests for 512 kb

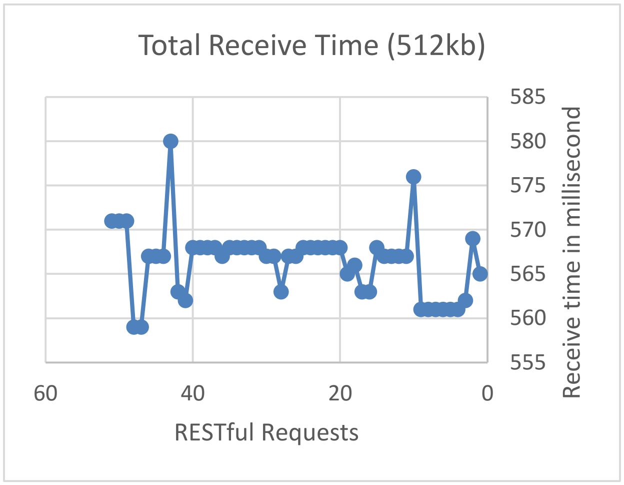


Figure (6.13) Receive time of RESTful requests for 512kb

6.1.7 Experiment for 1 Mb data size

In this experiment we send the request with parameter size*1m so the service get the parameter and response with JSON data the size of it is 1 Mb.

The link will be like the following:

http://masterdegree.Azurewebsites.net/ bufferOverFlowAttackApi/getAllCategories/size*1m/

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

- ☒ **Results**

After analyze the excel file that get it from HAR file ,we use the average method to get the average of total response time in millisecond since we note that the average response time is 641.01 ms and the average of total receive time is 888.82 ms .

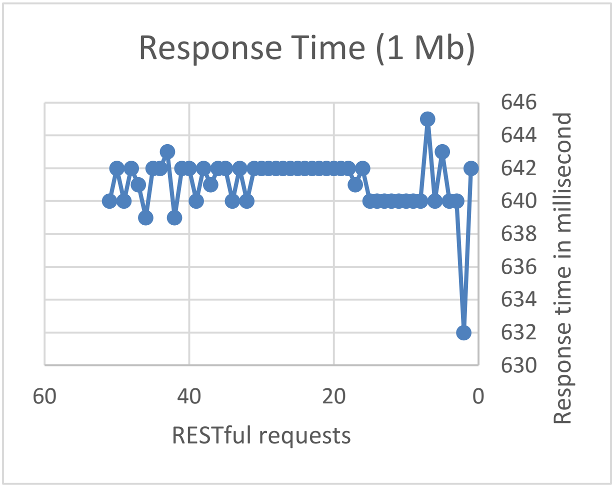


Figure (6.16) Response time of RESTful requests for 1 Mb

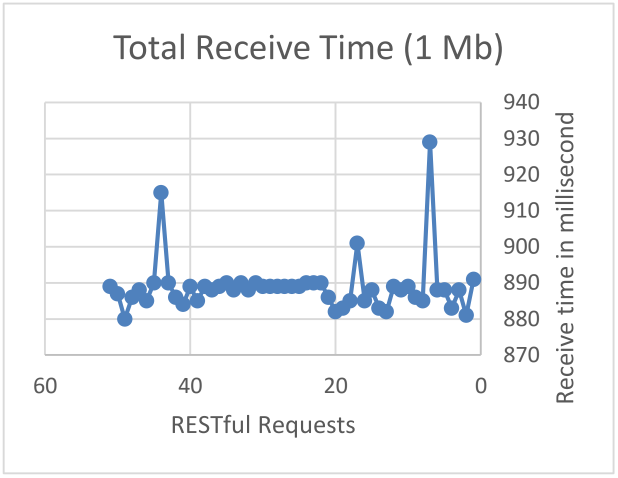


Figure (6.15) Receive time of RESTful requests for 1 Mb

6.1.8 Experiment for 2 Mb data size

In this experiment we send the request with parameter size*2m so the service get the parameter and response with JSON data the size of it is 2 Mb.

The link is:

http://masterdegree.Azurewebsites.net/ bufferOverFlowAttackApi/getAllCategories/size*2m/

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

- **Results**

After analyze the excel file that get it from HAR file ,we use the average method to get the average of total response time in millisecond since we note that the average response time is 1279.86 ms and the average of total receive time is 1528.04 ms .

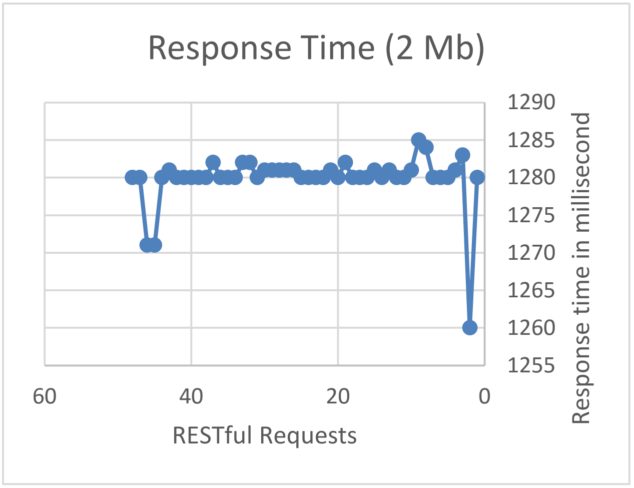


Figure (6.17) Response time of RESTful requests for 2 Mb

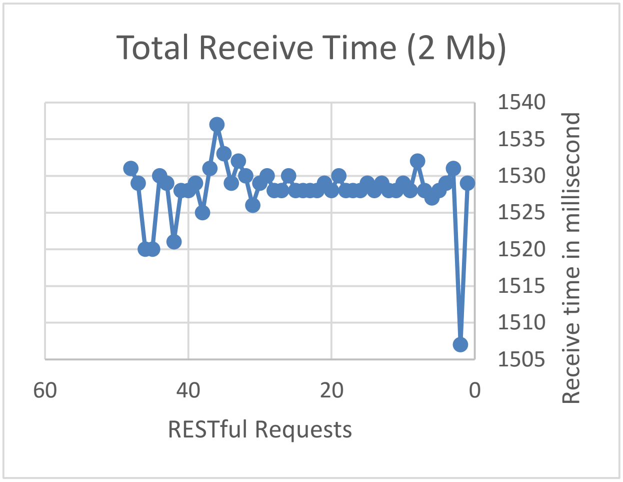


Figure (6.18) Receive time of RESTful requests for 2 Mb

6.1.9 Experiment for 3 Mb data size

In this experiment we send the request with parameter size*3m so the service get the parameter and response with JSON data the size of it is 3 Mb.

The link is:

http://masterdegree.Azurewebsites.net/ bufferOverFlowAttackApi/getAllCategories/size*3m/

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

☒ **Results**

After analyze the excel file that get it from HAR file, we use the average method to get the average of total response time in millisecond since we note that the average response time is 1920.6 ms and the average of total receive time is 2167.6 ms .

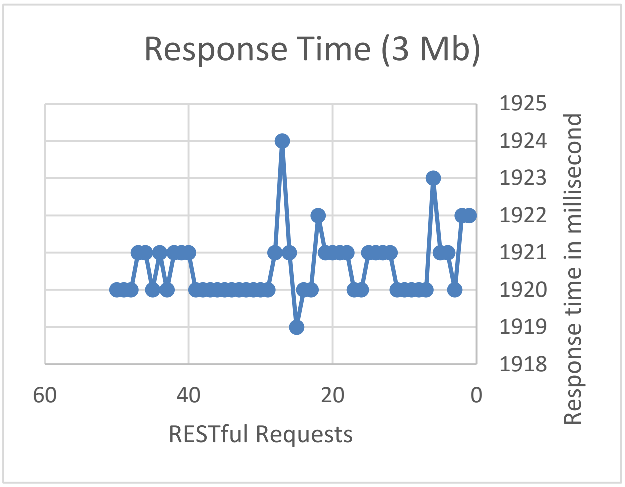


Figure (6.19) Response time of RESTful requests for 3 Mb

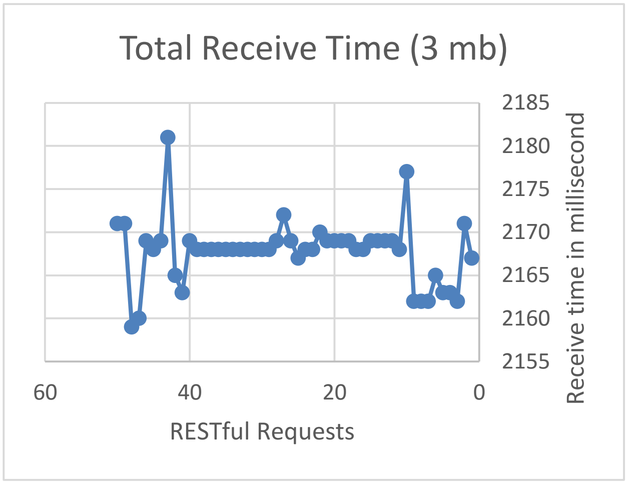


Figure (6.20) Receive time of RESTful requests for 3 Mb

6.1.10 Experiment for 4 Mb data size

In this experiment we send the request with parameter size*4m so the service get the parameter and response with JSON data the size of it is 4 Mb.

The link is:

http://masterdegree.Azurewebsites.net/ bufferOverFlowAttackApi/getAllCategories/size*4m/

Using Firebug tools and netExport tools we save all details of requests into HAR file which we convert it into Excel file using har2csv tools and processing the excel file to find the following results.

- **Results**

After analyze the excel file that get it from HAR file, we use the average method to get the average of total response time in millisecond since we note that the average response time is 2560.9 ms and the average of total receive time is 2808.08 ms .

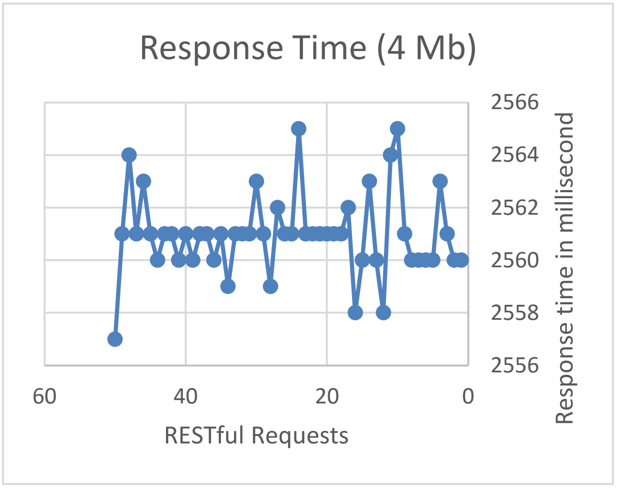


Figure (6.21) Response time of RESTful requests for 4 Mb

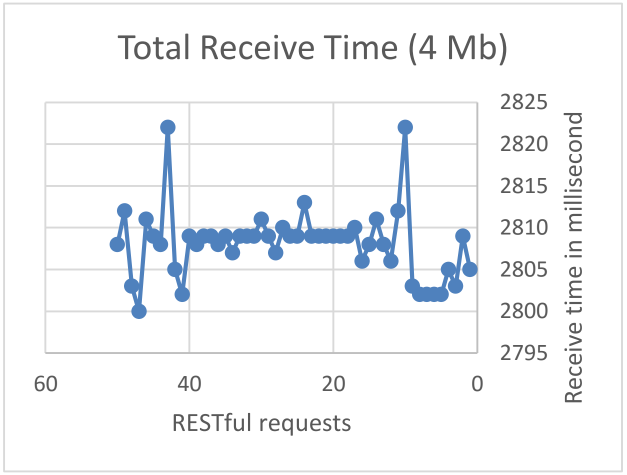


Figure 6.22) Receive time of RESTful requests for 4 Mb

6.1.11 Experiment for 5 Mb data size

In this experiment we send the request with parameter size*5m so the service get the parameter and response with JSON data the size of it is 5 Mb.

The link is:

http://masterdegree.Azurewebsites.net/ bufferOverFlowAttackApi/getAllCategories/size*5m/

We will convert the HAR file to Excel file with same step in previous experiment.

- **Results**

After analyze the excel file that get it from HAR file ,we use the average method to get the average of total response time in millisecond since we note that the average response time is 3201.3 ms and the average of total receive time is 3449.8 ms .

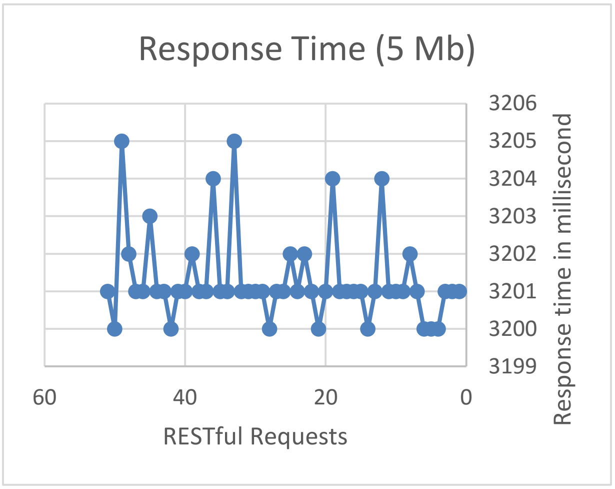


Figure (6.23) Response time of RESTful requests for 5 Mb

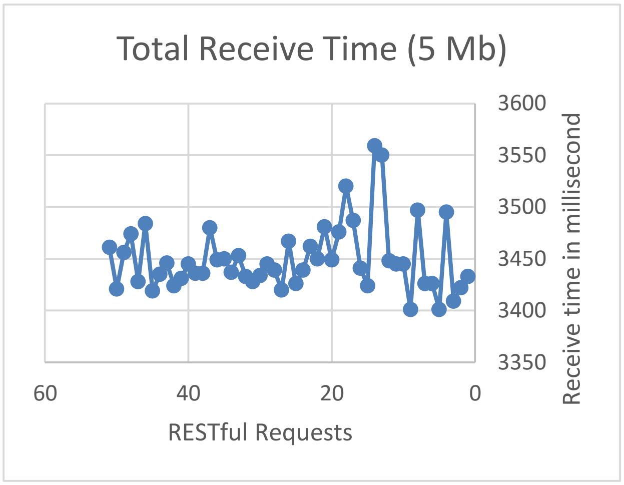
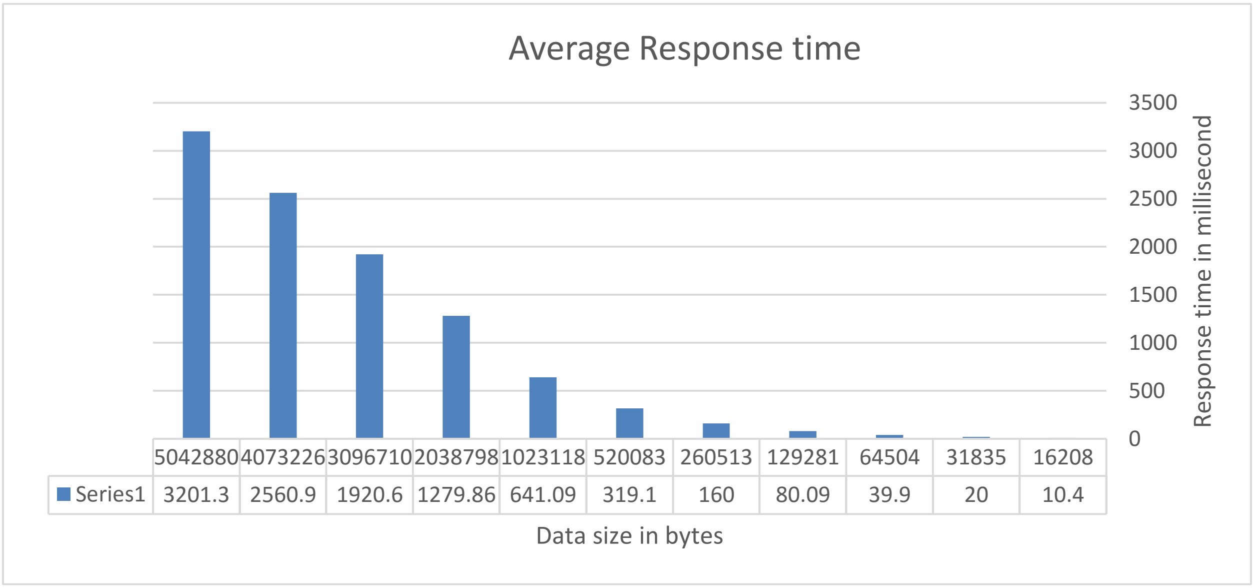


Figure 6.24) Receive time of RESTful requests for 5 Mb

6.1.12 Results & Justification for Experiment part I

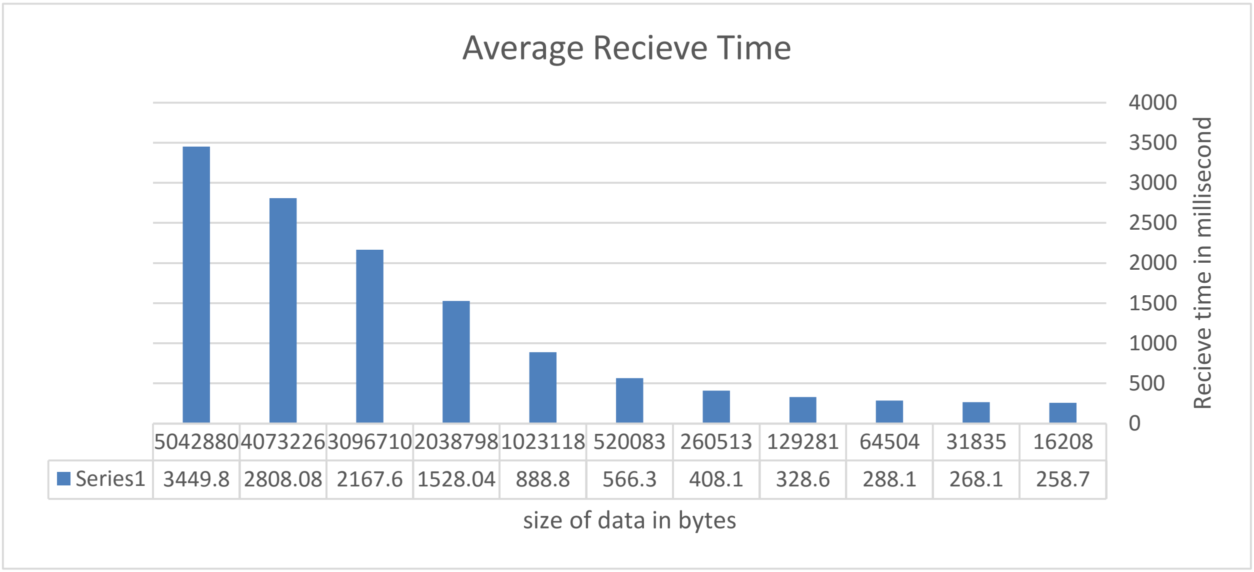
Our results depend on response time and receive time that consists of both time response plus waiting time. We note that when the size of retrieve data increase then the response time will increase. Figure 6.25 shows the response time of different data size.



	5042880	4073226	3096710	2038798	1023118	520083	260513	129281	64504	31835	16208
■ Series1	3201.3	2560.9	1920.6	1279.86	641.09	319.1	160	80.09	39.9	20	10.4

Figure (6.25) Average Response time for different size of response data

In our experiment we use lightly weight of JSON response data, thus every time we double the size of retrieve data, we note the response time will double also, and there is waiting time is as constant partially equal 248 milliseconds for all experiment used lightly weight, figure 5.16 shows the average total receives time for different size of data, as we know this time consists of time response plus waiting time.



	5042880	4073226	3096710	2038798	1023118	520083	260513	129281	64504	31835	16208
■ Series1	3449.8	2808.08	2167.6	1528.04	888.8	566.3	408.1	328.6	288.1	268.1	258.7

Figure (6.26) Average Receive time for different size of response data

We conclude after performed experiments that the equation for response time is

$$\mathbf{T_{re} = (X/T_c) *10}$$

Where Tre is response time, X is data size and $\mathbf{T_c}$ is a constant equal 16000 according to our experiment and Azure environment which may change on another cloud environment when we calculate the total receive time we need to increase the waiting time which as we mentioned is as constant in our experiment, thus the equation for receive time is

$$\mathbf{T_{rec} = T_{re} + T_W}$$

Where Trec is the receive time, Tre the response time, and **Tw** is the waiting time (Allowance time) for used server where equal in our experiment to 248 ms.

After we have the receive time equation now we can substitute in Hussam's algorithms to evaluate our work and the correctly of equation.

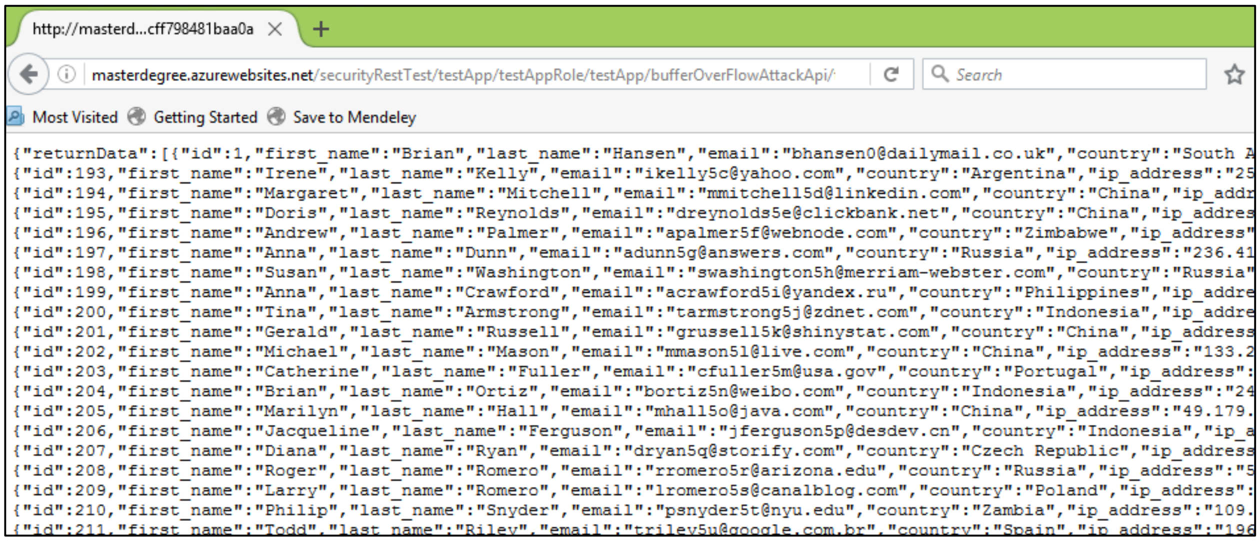
6.2 Experiment Part II

In this section we aim to evaluate our approach using our enhanced algorithms, thus we substitute the next request time as constant in our equation Trec that represents the receive time.

In the algorithm we use our RESTful service getAllCategories and perform the experiment using different size of retrieve data.

6.2.1 Evaluate equation using 512 kb

In this section we use 512 kb of retrieve data from RESTful service and depend on our equation we will substitute the next request time with our equation, then send 5 request to service ,for first request the algorithm send the required data and we noted that for all requests the algorithm will retrieve data and serve all requests that's because the next request time according to our equation will increased the current time about 0.568 second , this is very small time , Figure 6.27 shows the response data from request .


http://masterd...cff798481baa0a
masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/
Search
Most Visited
Getting Started
Save to Mendeley

```
{"returnData":[{"id":1,"first_name":"Brian","last_name":"Hansen","email":"bhansen0@dailymail.co.uk","country":"South A
{"id":193,"first_name":"Irene","last_name":"Kelly","email":"ikelly5c@yahoo.com","country":"Argentina","ip_address":"25
{"id":194,"first_name":"Margaret","last_name":"Mitchell","email":"mmitchell5d@linkedin.com","country":"China","ip_addr
{"id":195,"first_name":"Doris","last_name":"Reynolds","email":"dreynolds5e@clickbank.net","country":"China","ip_addres
{"id":196,"first_name":"Andrew","last_name":"Palmer","email":"apalmer5f@webnode.com","country":"Zimbabwe","ip_address"
{"id":197,"first_name":"Anna","last_name":"Dunn","email":"adunn5g@answers.com","country":"Russia","ip_address":"236.41
{"id":198,"first_name":"Susan","last_name":"Washington","email":"swashington5h@merriam-webster.com","country":"Russia"
{"id":199,"first_name":"Anna","last_name":"Crawford","email":"acrawford5i@yandex.ru","country":"Philippines","ip_addre
{"id":200,"first_name":"Tina","last_name":"Armstrong","email":"tarmstrong5j@zdnet.com","country":"Indonesia","ip_addre
{"id":201,"first_name":"Gerald","last_name":"Russell","email":"grussell5k@shinystat.com","country":"China","ip_address
{"id":202,"first_name":"Michael","last_name":"Mason","email":"mmason5l@live.com","country":"China","ip_address":"133.2
{"id":203,"first_name":"Catherine","last_name":"Fuller","email":"cfuller5m@usa.gov","country":"Portugal","ip_address":
{"id":204,"first_name":"Brian","last_name":"Ortiz","email":"bortiz5n@weibo.com","country":"Indonesia","ip_address":"24
{"id":205,"first_name":"Marilyn","last_name":"Hall","email":"mhall5o@java.com","country":"China","ip_address":"49.179.
{"id":206,"first_name":"Jacqueline","last_name":"Ferguson","email":"jferguson5p@desdev.cn","country":"Indonesia","ip_a
{"id":207,"first_name":"Diana","last_name":"Ryan","email":"dryan5q@storify.com","country":"Czech Republic","ip_address
{"id":208,"first_name":"Roger","last_name":"Romero","email":"rromero5r@arizona.edu","country":"Russia","ip_address":"5
{"id":209,"first_name":"Larry","last_name":"Romero","email":"lromero5s@canalblog.com","country":"Poland","ip_address":
{"id":210,"first_name":"Philip","last_name":"Snyder","email":"psnyder5t@nyu.edu","country":"Zambia","ip_address":"109.
{"id":211,"first_name":"Todd","last_name":"Riley","email":"triley5u@google.com.br","country":"Spain","ip_address":"196
```

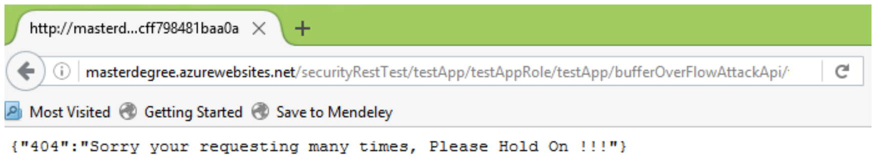
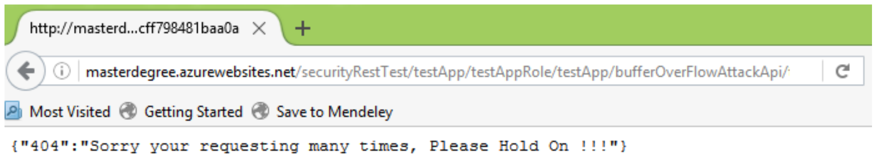
Figure (6.27) Output data for RESTful request for 512 Kb

6.2.2 Evaluate equation using 2 M

In this section we use 2 M of retrieve data from RESTful service and depend on our equation we will substitute the next request time with our equation, then send 5 request to service, we note according to our equation is the next request time response must be current time + 1558 ms.

When we send 5 requests to our enhanced algorithm the results will be as shown in table 6.2

Table (6.2) Results of evaluate equation using 2 M

Number of request	**Response message**
Request 1	JSON data from service (response time 1360.4ms)
Request 2	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ {"404":"Sorry your requesting many times, Please Hold On !!!"}
Request 3	JSON data from service (response time 1320.6 ms)
Request 4	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ {"404":"Sorry your requesting many times, Please Hold On !!!"}
Request 5	JSON data from service (response time 1516.3 ms)

We note that the average response time after apply our experiments on enhanced algorithm is 1399.1ms, and according to our equation the response time must be 1310.7 ms.

We used the following equation to calculate the percent of error:

$$\textbf{percent error} = |(V_{true} - V_{used})/V_{true}| * 100$$

Where: V_{true} is the true value
V_{used} is the value used

= | (1310.7-1399.1)/1310.7|*100

= **6.7445%**

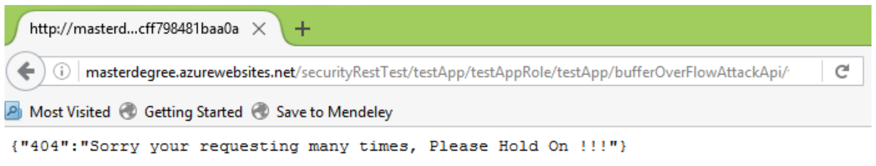
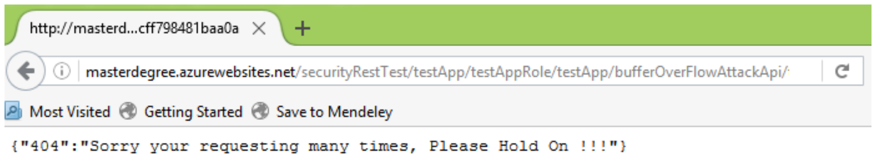
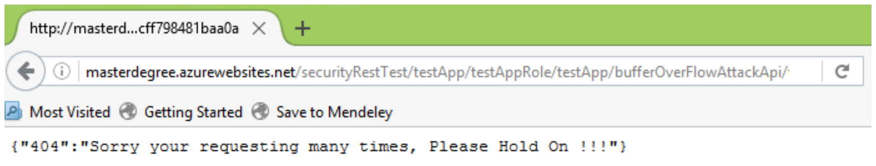
After found the percent error we can said that the accuracy of our equation is 93.255 % the precision is very good as it's affected with data size which retrieved from our services.

6.2.3 Evaluate equation using 3 M

In this section we use 3 M of retrieve data from RESTful service and depend on our equation we will substitute the next request time with our equation, then send 5 request to service, we note according to our equation is the next request time response must be current time + 2167.6 ms.

When we send 5 requests to our enhanced algorithm the results will be as shown in table 6.3

Table (6.3) Results of evaluate equation using 3 M

Number of request	**Response message**
Request 1	JSON data from service (response time 1950.4ms)
Request 2	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 3	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 4	JSON data from service (response time 1943.3 ms)
Request 5	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`

We note that the average response time after apply our experiments on enhanced algorithm is 1946.8 ms, and according to our equation the response time must be 1919.6 ms.

We used the following equation to calculate the percent of error:

$$\textbf{percent error} = |(V_{true} - V_{used})/V_{true}| * 100$$

Where: V_{true} is the true value
V_{used} is the value used

$$= |(1919.6 - 1946.8)/ 1919.6|*100$$
$$= \mathbf{1.417\ \%}$$

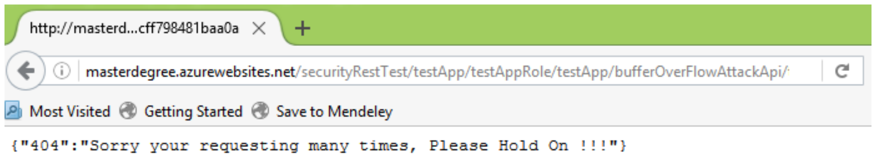
After found the percent error we can said that the accuracy of our equation is 98.58 % the precision is very good as it's affected with data size which we note when the size of data increased the accuracy of our equation will increase.

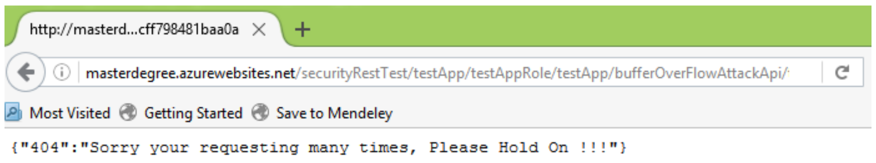
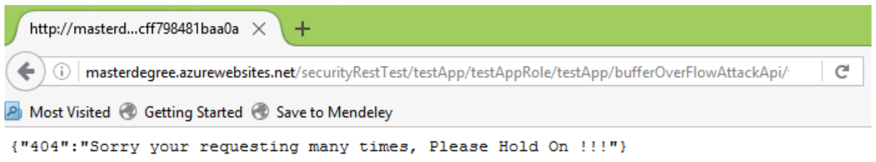
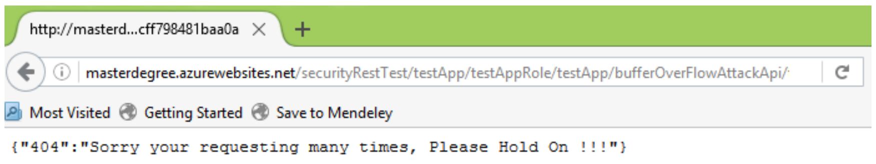
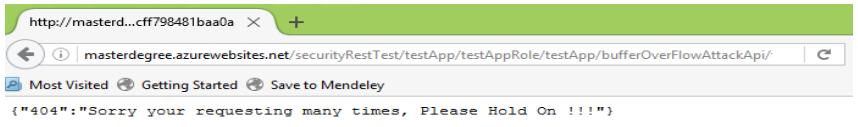
6.2.4 Evaluate equation using 4 M

In this section we use 4 M of retrieve data from RESTful service and depend on our equation we will substitute the next request time with our equation, then send 10 request to service, we note according to our equation is the next request time response must be current time + 2621.4 ms .

When we send 10 requests to our enhanced algorithm the results will be as shown in table 6.4

Table (6.4) Results of evaluate equation using 4 M

Number of request	**Response message**
Request 1	JSON data from service (response time 2690.3 ms)
Request 2	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`

Number of request	**Response message**
Request 3	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 4	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 5	JSON data from service (response time 2650.7 ms)
Request 7	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 8	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`

We note that the average response time after apply our experiments on enhanced algorithm is 2670.5 ms, and according to our equation the response time must be 2621.4 ms.

We used the following equation to calculate the percent of error:

$$\textbf{percent error} = |\,(V_{true} - V_{used})/V_{true}|\ * 100$$

Where: V_{true} is the true value
V_{used} is the value used

= | (2621.4 - 2670.5)/ 2621.4 |*100
= **1.87%**

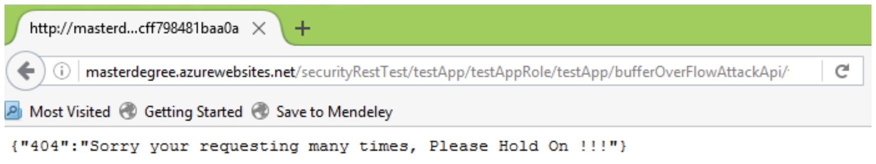
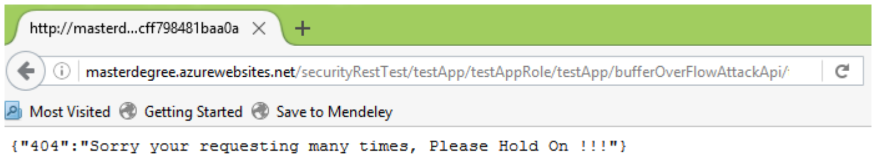
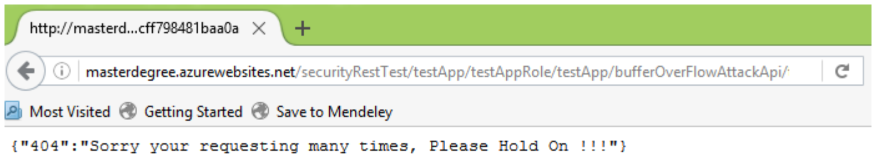
After found the percent error we can said that the accuracy of our equation is 98.10 % as we noted when the size of data increased the accuracy of our equation will increase.

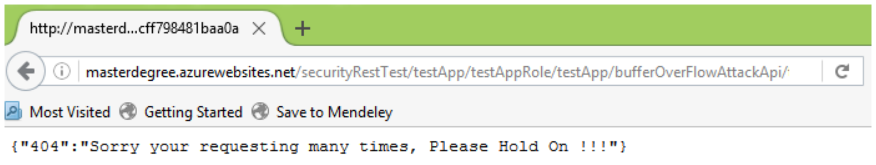
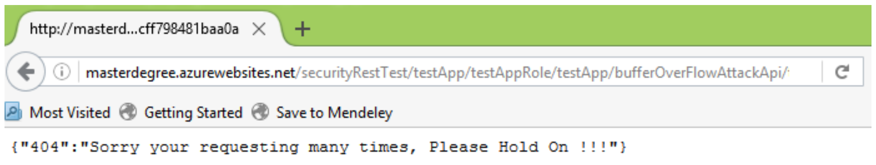
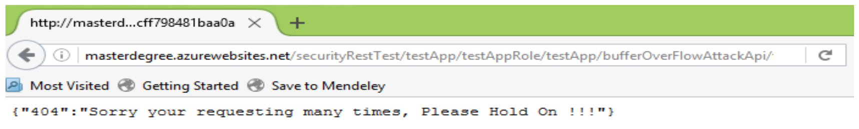
6.2.5 Evaluate equation using 5 M

In this section we use 5 M of retrieve data from RESTful service and depend on our equation we will substitute the next request time with our equation, then send 10 request to service, we note according to our equation is the next request time response must be current time + 3449.8 ms .

When we send 5 requests to our enhanced algorithm the results will be as shown in table 6.5

Table (6.5) Results of evaluate equation using 5 M

Number of request	**Response message**
Request 1	JSON data from service (response time 3250.3 ms)
Request 2	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 3	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 4	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ `{"404":"Sorry your requesting many times, Please Hold On !!!"}`

Number of request	**Response message**
Request 5	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 6	JSON data from service (response time 3242.5 ms)
Request 7	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`
Request 8	Refused get data  http://masterd...cff798481baa0a masterdegree.azurewebsites.net/securityRestTest/testApp/testAppRole/testApp/bufferOverFlowAttackApi/ Most Visited Getting Started Save to Mendeley `{"404":"Sorry your requesting many times, Please Hold On !!!"}`

We note that the average response time after apply our experiments on enhanced algorithm is 3246.4 ms, and according to our equation the response time must be 3201.8 ms.

We used the following equation to calculate the percent of error:

$$\textbf{percent error} = |(V_{true} - V_{used})/V_{true}| * 100$$

Where: V_{true} is the true value
V_{used} is the value used

$$= |(3201.8 - 3246.4)/ 3201.8|*100$$
$$= \mathbf{1.393\%}$$

After found the percent error we can said that the accuracy of our equation is 98.60 % as we noted when the size of data increased the accuracy of our equation will increase.

6.2.6 Results & Justification for Experiment part II

We note when apply our equation on Hussam's algorithm and enhancing it, the client cants repeated malicious requests because the algorithm will serve the request and check if the service complete the work, then take another request to serve it. When using this algorithm, we will reduce the chance of DOS attack on our server and reduce the money cost from each request which take bandwidth. In our evaluation experiments we noted that the algorithm will perform well when the data size increased and the accuracy of our equation is very closely for large size of data, Figure 6.28 shows the accuracy of response time for different size of data.

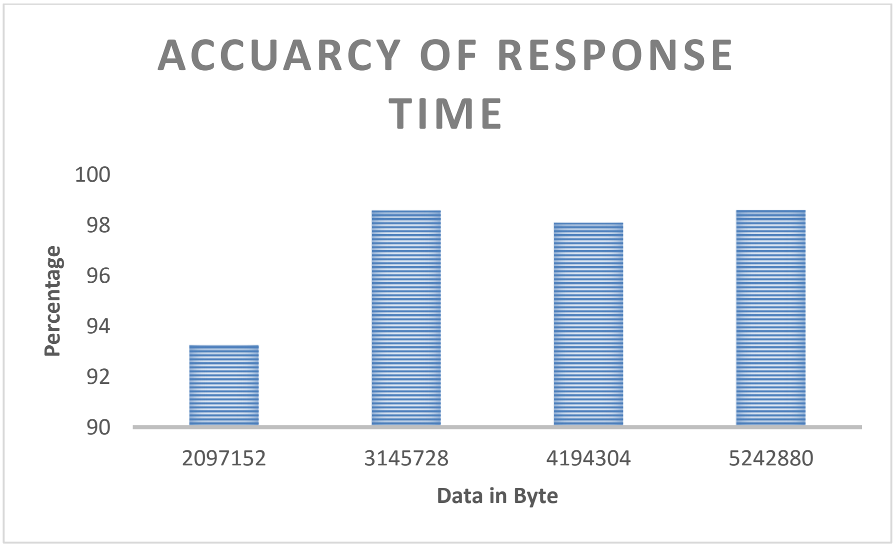


Figure (6.28) Accuracy of response time from enhanced algorithm

6.3 Summary and Discussion

In this chapter the experiments were presented and discussed in details. The experiments were done using Windows Azure environment. The experiments consist of two parts. Firstly, we presented a study of response time for RESTful requests using different sizes of retrieved data. In this part we get the following equation to estimate the response time:

Tre = (X/ $\mathbf{T_c}$) *10

$\mathbf{T_c}$ equal 16000 is appeared according to results of our experiment, its specific for windows azure and may change when we do the experiment on another cloud environment

The second part of this chapter contains the experiments which test and evaluate the results and equations of the proposed approach. We calculated the accuracy of our equations using two, three, and five megabyte of retrieved data where we found the accuracy increased when the size of data increased to be 98.60 % when the size of data is five megabytes.

Table 6.6 discussed the difference of our approach and other related work in preventing DOS attack on web services

Table (6.6) Summarization of most related work

Author	**Related work: Problem and proposed solution**	**Our Approach**
H. Elkurd and T. Barhoom	Proposed an algorithm to prevent buffer overflow DOS Attack in RESTful Services by comparing the difference between current time for RESTful request and the time for previous RESTful request with the current process computing time.	In our approach, we aim to enhance this algorithm by substitute the constant for next request time by equation to estimate the expected time for next RESTful request.
Irfan siddavatam	Propose comprehensive set of test mechanism to detect attack on web services. he set guard time period to limit request in frame by admin configuration and preventing buffer overflow attack	In our approach we add dynamic different guard time for each action, thus computing time is different from one resource to another according to data size.

Author	**Related work: Problem and proposed solution**	**Our Approach**
Zhang Chao-yang	Proposed to prevent DOS attacks by adding certification system defense that define the identity for each client, this limit the probability of attacks outside the defined clients.	We found this solution is costly, in our approach we can define the identity of authorized clients as condition to generate new tokens.
Gabriel	Provide a security protocol to make message security implementation by encrypt message exchange between the provider and client throw new encrypted headers in the http request and response in order to meet RESTful principles.	While the idea is novel and provide security for RESTful services equivalent to WS-Security but it always need to overload request and response with extra headers and this is drawback. Our approach aims to reduce DOS attack by put guard time to reduce consuming the resources by malicious requests.
Jiang, Lee and Songlin	Present one such a largescale longitudinal analysis of publicly available web services of SOAP-based and RESTful types	We use this study to be aware about the quality of service after apply our approach, where the response time one of the important factors of QoS.

Chapter 7
Conclusion and Future work

Chapter 7
Conclusion and Future Work

In this research we have designed an approach to get next time for RESTful request and enhanced algorithm to reduce DOS attack on RESTful services, the approach consists of two main phases: the first one is to collect and prepare the data set, and the second one for processing results and enhance the algorithm to evaluate the results.

We conclude after performed experiments that the equation for response time is

$Tre = (X/16000) *10$

Where Tre is response time and X is data size. The number of 16000 is appeared according to results of our experiment, its specific for windows azure and may change when we do the experiment on another cloud environment. We calculate the total receive time we need to increase the waiting time which as we mentioned is as constant in our experiment, thus the equation for receive time is $Trec = Tre + W$

Where Trec is the receive time, Tre the response time, and W is the waiting time for used server where equal in our experiment to 248 ms

The second part contains the experiments which test and evaluate the results and equations of the proposed approach. The test gets the expected result, therefore while we are running request on the requested resource over REST API the new requests will be prevented until the defined next request time is occurred, thus we calculated the error percent of our equation to be in range 6.7% - 1.3% that means the accuracy of our equation from 93.3% to 98.7% with average 97.31 %.

As a future work, the research approach can be updated through adding the following:

- Find an effective way to prevent the attack without violating REST principles like not save any history from last request and depend on current state of request and enhance the approach to prevent the slow connection attack.
- Test the proposed algorithm on different other real cloud like amazon to get general equation that we can apply it on different environment.

The Reference List

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., … Zaharia, M. (2009). Above the Clouds : A Berkeley View of Cloud Computing Cloud Computing : An Old Idea Whose Time Has (Finally) Come. *UC Berkeley Reliable Adaptive Distributed Systems Laboratory Http://radlab.cs.berkeley.edu/*, 7–13.

Avram, A. (2011). Is REST Successful in the Enterprise? Retrieved April 9, 2015, from http://www.infoq.com/news/2011/06/Is-REST-Successful

Backere, F. De, Hanssens, B., Heynssens, R., Houthooft, R., Zuliani, A., Verstichel, S., … Turck, F. De. (2014). Design of a Security Mechanism for RESTful Web Service Communication through Mobile Clients. *IEEE*.

Buyya, R., Yeo, C. S., & Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. *Proceedings - 10th IEEE International Conference on High Performance Computing and Communications, HPCC 2008*, 5–13. http://doi.org/10.1109/HPCC.2008.172

Cheng, Y. C., Laih, C. S., Lai, G. H., Chen, C. M., & Chen, T. (2008). Defending on-line web application security with user-behavior surveillance. *ARES 2008 - 3rd International Conference on Availability, Security, and Reliability, Proceedings*, 410–415. http://doi.org/10.1109/ARES.2008.127

Chepaitis, B. A. (2003). Learning FileZilla in 5 Minutes. *FileZilla*, pp. 1–7. Retrieved from http://www.framasoft.net/article1941.html

Collier, M., & Shahan, R. (2015). *Fundamentals of Azure Microsoft Azure Essentials*. Microsoft Press A Division of Microsoft Corporation One Microsoft Way Redmond, Washington 98052-6399 Copyright.

Cotroneo, D., Peluso, L., Romano, S. P., Ventre, G., & Napoli, V. C. (2002). An Active Security Protocol against DoS attacks. *Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)*.

Elkurd, H. A., & Barhoom, T. S. (2015). Preventing Buffer Overflow DOS Attack in Restful Services. *International Journal of Information and Communication Technology Research*, *5*(1), 13–15.

Fielding, R. T. (2015). The REST Architectural Style List - Yahoo Groups. Retrieved April 9, 2015, from https://groups.yahoo.com/neo/groups/rest-discuss/conversations/topics/5841

Fouladi, R. F., Seifpoor, T., Anarim, E., & Engineering, E. (n.d.). Frequency Characteristics ofDoS and DDoS Attacks, 6–9.

Google Apps Tool. (2014). HAR Analyzer. Retrieved March 30, 2016, from https://toolbox.googleapps.com/apps/har_analyzer/

Heller, M. (2007). REST and CRUD: the Impedance Mismatch. *Developer World*. Retrieved from http://www.infoworld.com/d/developer-world/rest-and-crud-impedance-mismatch-927

Jiang, W., Lee, D., & Hu, S. (2012). Large-scale longitudinal analysis of SOAP-based and RESTful web services. *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, 218–225. http://doi.org/10.1109/ICWS.2012.45

M. Jones, D. H. (2012). The OAuth 2.0 Authorization Framework: Bearer Token Usage. Retrieved June 6, 2015, from http://www.hjp.at/doc/rfc/rfc6750.html#sec_4

Marinos, A., & Briscoe, G. (2009). Community Cloud Computing. *arXiv:0907.2485v3 [cs.NI] 12 Oct*.

Nandgaonkar, S. V, & Raut, P. a B. (2014). A Comprehensive Study on Cloud Computing. *International Journal of Computer Science and Mobile Computing*, *3*(4), 733–738.

Park, J., Iwai, K., Tanaka, H., & Kurokawa, T. (2014). Analysis of Slow Read DoS attack. In *Information Theory and its Applications (ISITA), International Symposium on* (pp. 60–64). Melbourne, VIC: IEEE.

Pautasso, C., Alarcon, R., & Wilde, E. (2014). *REST: Advanced Research Topics and Practical Applications*. Retrieved from http://www.springer.com/engineering/signals/book/978-1-4614-9298-6

Preimesberger, C. (2014). DDoS Attack Volume Escalates as New Methods Emerge. Retrieved April 9, 2015, from http://www.eweek.com/security/slideshows/ddos-attack-volume-escalates-as-new-methods-emerge.html

Richardson, L., & Ruby, S. (2008). *RESTful Web Services*. *Vasa*. http://doi.org/10.1109/MIC.2008.130

Road, S., & Kingdom, U. (2008). Turning Software into a Service. *IEEE Computer Society*, *44*(October), 38–44.

Rodriguez, A. (2015, February 9). RESTful Web services: The basics. Retrieved from http://www.ibm.com/developerworks/library/ws-restful/

Rouached, M. (2013). RESTful Web Services for High Speed Intrusion Detection Systems. *IEEE 20th International Conference on Web Services*. http://doi.org/10.1109/ICWS.2013.92

Saeed Araban, L. S. (2004). Quality of Service for Web Services. *IBM Developerworks Library*, (January), 1–9. Retrieved from http://www.wseas.us/e-library/conferences/austria2004/papers/482-350.pdf

Sajid, M. (2013). Cloud Computing: Issues & Challenges. *International Conference on Cloud, Big Data and Trust 2013, Nov 13-15, RGPV*.

Sandy Mappic. (2012). How Fast are your Web Services? Retrieved June 4, 2016, from https://blog.appdynamics.com/cloud/how-fast-are-your-web-services/

Sarna, D. E. Y. (n.d.). *Implementing and Developing Cloud Computing Applications*. Retrieved from https://books.google.com/books?hl=ar&lr=&id=n1zRBQAAQBAJ&pgis=1

Serme, G., De Oliveira, A. S., Massiera, J., & Roudier, Y. (2012). Enabling message security for RESTful services. *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, 114–121. http://doi.org/10.1109/ICWS.2012.94

Services, W., & Architecture, C. (2001). Web Services Conceptual Architecture (WSCA 1.0). *Architecture*, *5*(May), 6–7. Retrieved from http://www.csd.uoc.gr/~hy565/newpage/docs/pdfs/papers/wsca.pdf

Siddavatam, I., & Jayant Gadge. (2008). Comprehensive test mechanism to detect attack on web services. *Proceedings of the 2008 16th International Conference on Networks, ICON 2008*. http://doi.org/10.1109/ICON.2008.4772620

Team, M. (2015). Mashape Docs - Overview. Retrieved March 31, 2016, from http://docs.mashape.com/overview

Tulloch, M. (2013). *Introducing Windows Azure*. (A. Hamilton, Ed.). Microsoft Press A Division of Microsoft Corporation One Microsoft Way Redmond, Washington 98052-6399 Copyright. http://doi.org/10.1201/1086/43311.9.4.20000910/31367.7

Zhang, C. Y. (2011). DOS attack analysis and study of new measures to prevent. *Proceedings - 2011 International Conference on Intelligence Science and Information Engineering, ISIE 2011*, 426–429. http://doi.org/10.1109/ISIE.2011.66

Zheng, Z., Zhang, Y., & Lyu, M. R. (2014). Investigating QoS of real-world web services. *IEEE Transactions on Services Computing*, *7*(1), 32–39. http://doi.org/10.1109/TSC.2012.34