COMBINING CACHING WITH A CLOUD HOSTED PROXY TO SUPPORT MOBILE CONSUMERS OF RESTFUL SERVICES

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Masters of Science

In the Department of Computer Science

University of Saskatchewan

Saskatoon

By

SHOMOYITA JAMAL

ABSTRACT

There are numerous problems to be addressed when connecting mobile clients (e.g. smartphones and tablet devices) with Web services. These devices consume Web services via wireless channels; and as a result, developers and researchers are investigating different approaches to address challenges related to network fluctuation, latency, and low bandwidth. In addition, most of these devices have limited capabilities in terms of information processing and resource storage. This research focuses on enabling mobile devices for consuming RESTful Web services efficiently.

The aforementioned problems of network instability are addressed in this research by proposing and implementing a cloud centric proxy server architecture; which is based on mirroring resources. The mirroring of the Web server's resources on the mobile device and the proposed proxy server is achieved by exploring caching techniques. Furthermore, an evaluation is done to determine what kind of components and architecture is required for supporting resource constraint mobile devices like smartphones and tablets while connecting them with RESTful systems. By linking the caching components of the mobile devices with a cloud-hosted proxy server, it becomes possible to share caches and achieve significant performance boost for mobile consumers of the RESTful Web services.

ACKNOWLEDGMENTS

First of all I would like to express my sincere gratitude to my supervisor, Dr. Ralph Deters for his advice and assistance throughout my entire duration of study. Also, my sincere thanks to the members of my advisory committee: Dr. Julita Vassileva, Dr. Chanchal Roy and Dr. A. Wahid Khan for their valuable advice and suggestions. Furthermore, I would like to thank Mr. Andrew Kostiuk and TRLabs for their support. Also, I would like to appreciate the assistance of Ms. Jan Thompson and Ms. Gwen Lancaster, at the Department of Computer Science, University of Saskatchewan, for their support and generosity.

Thanks to all students of the Multi-Agent Distributed Mobile and Ubiquitous Computing (MADMUC) Lab for their friendship.

Finally, I would like to thank my husband, Mohammad Asif Ashraf Khan; my mother, Shyamoli Akbar; my father A. F. M. Jamaluddin and my entire family for the unconditional love and support throughout this period.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CPU	Central Processing Unit
EC2	Amazon Elastic Cloud Computing
GAE	Google App Engine
HATEOAS	Hypermedia as the Engine of Application State
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IP	Internet Protocol
JSON	JavaScript Object Notation
LDD	Location Dependent Data
MR	Mixed Reality
MSS	Mobile Support Station
OS	Operating System
OTP	Open Telecom Platform
PaaS	Platform as a Service
POX	Plain Old XML
RAM	Random Access Memory
REST	Representational State Transfer
ROA	Resource Oriented Architecture
SaaS	Software as a Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WADL	Web Application Description Language
WS	Web Services
WS*	Web Services
WSDL	Web Services Description Language
WWW	World Wide Web
XML	Extensible Markup Language

CHAPTER 1
INTRODUCTION

In the last decade, Web Services (WS) [58] have emerged as the standard for building open distributed system. WS provide an open and platform independent approach to connect various components within and across physical and organizational borders. At the same time, we have witnessed significant growth within mobile handsets worldwide. 476.5 million mobile devices were sold in the fourth quarter of 2011 worldwide [20][21] and every third mobile device was sold was a smartphone. The value of the smartphone grows exponentially by its ability to connect to external WS. This leads us to the question of how to connect this resource constrained mobile device to the rich pool of different WS. Figure 1-1 and Figure 1-2 show a tablet device and a smartphone device respectively.

Figure 1-1. BlackBerry Playboook tablet device [8]

Figure 1-2. BlackBerry smartphone device [34]

Mobile devices such as smartphones and tablets are very heterogeneous. They have different operating systems, different hardware, and different form factors. As a result, developing a native app (application that is developed explicitly to use on a particular platform or device) means we are limited to building only platform specific applications. However, one way to deploy single code base applications on multiple mobile platforms to deal with the heterogeneity is by using the embedded browser pattern [31]. To achieve this, applications are often built using Web technology tools such as HTML5 and JavaScript to provide attractive user interfaces, client side functionalities, and inter component communications [31]. The flexibility of Web tools also affords developers the ability to wrap native codes in an embedded browser [31].

These mobile devices consume WS via wireless channels; and as a result, developers and researchers are exploring innovative ways to curtail the challenges of network fluctuation, latency, and low bandwidth. In addition, most of these mobile devices have limited capabilities in terms of information processing and resource storage. This research focuses on enabling mobile devices to consume RESTful [16] web services efficiently.

The main contribution of this research is the development of an architecture to address the above-mentioned problems of network instability. The architecture combines a local cache in the mobile device with a cloud hosted proxy server that works as a mediator or middleware. The mobile devices have caches for every individual resource which are mirrored resources. The middleware or proxy server is the intermediary between the REST Web server (which hosts the RESTful Web services) and mobile devices and also has caches for every single resource. This work also approaches the idea of sharing caches for multiple devices of a single user.

The rest of the thesis is structured as follows: Chapter 2 discusses the problem definition. Chapter 3 reviews research related to WS, consuming REST-WS by mobile devices, caching, proxy server and cloud computing. Chapter 4 discusses the proposed architecture. Chapter 5 details the prototypic implementation of the architecture and the programing techniques that are employed. Chapter 6 presents the experiments and evaluation of the work. And finally, Chapter 7 summarizes the research contribution and Chapter 8 concludes the paper with possible future work.

CHAPTER 2
PROBLEM DEFINITION

This research focuses on enabling resource constrained mobile devices to consume Web Services (WS) over wireless connections. Laptops, smartphones and tablets are emerging as popular consumers of WS (e.g. RESTful WS). The key question that needs to be addressed is: how to link these mobile devices wirelessly to the rich pool of different RESTful WS. Figure 2.1 shows the mobile devices consuming RESTful [16] WS in a Wi-Fi environment.

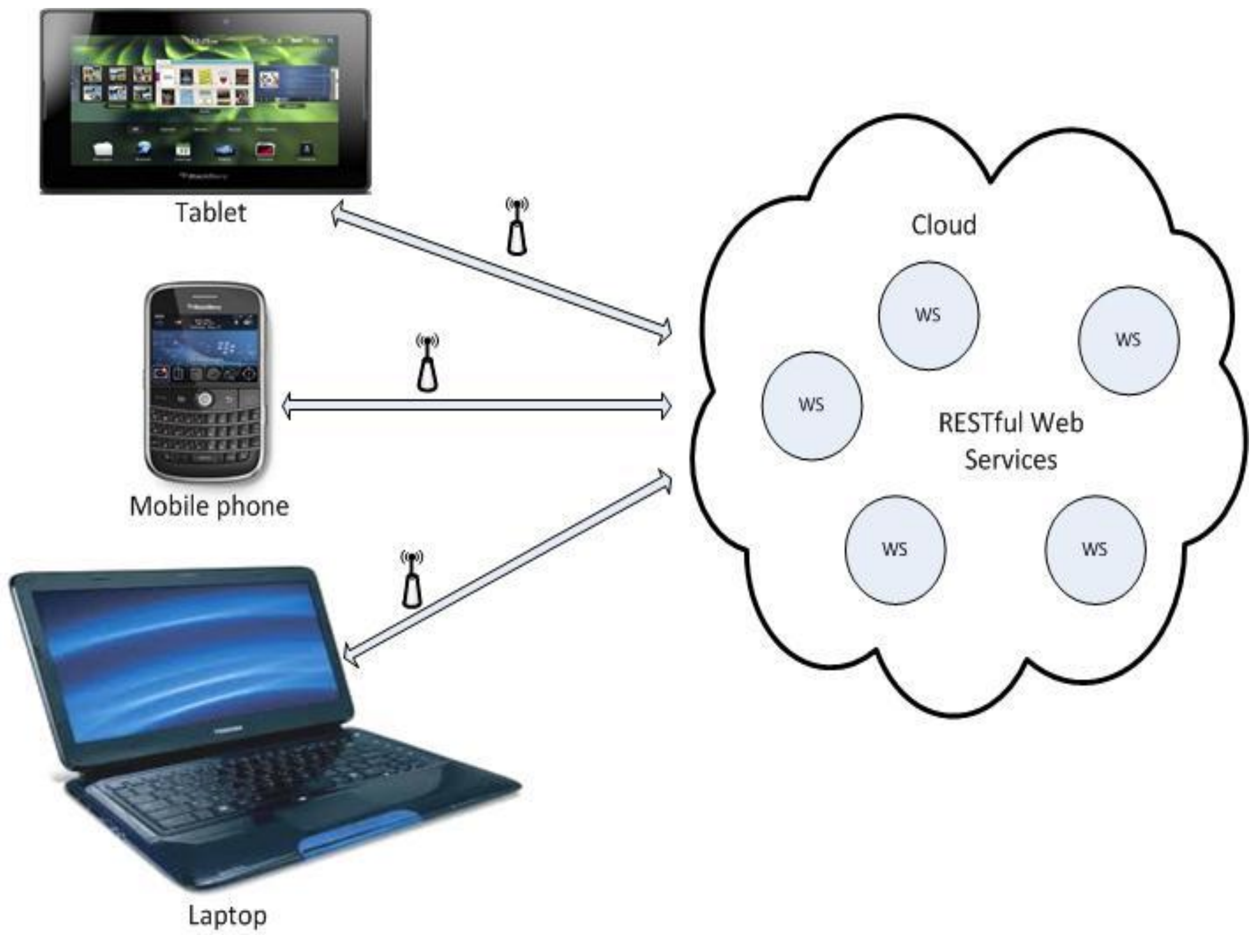


Figure 2-1. Mobile clients consuming RESTful WS in a Wi-Fi environment

To answer this question, four major problems must be addressed that are arising from wireless connectivity namely:

1. **Network loss/ Temporary network loss:** Mobile devices enforce location-dependent connectivity and often suffer network loss because of the mobility of the end user (or carrier). So the challenge is how to ensure seamless access to the services even when the device gets out of connection range.

2. **Latency:** The value of the mobile device is also determined by how fast the user can access the information. So another challenge is how latency can be decreased so that the information will always be available on the device whenever it is needed.

3. **Limited Resources:** The mobile device has limited resource. There is not a lot of CPU available because of energy limitation. How long a tablet device (e.g. iPad, PlayBook, and so on) can run also decides their demand and market value.

4. **Reduced bandwidth:** Mobile devices communicate over wireless channels which have limited bandwidth; and bandwidth is costly. Mobile consumers or the end users are billed on the amount of data transferred over the mobile network. Therefore the challenge is how the network traffic can be minimized so that the mobile consumers will have to pay less.

The goal of this research is to develop a cloud hosted proxy server platform and use caching on the mobile device to address the above mentioned challenges in order to facilitate the mobile devices to consume REST-WS efficiently. Figure 2.2 describes a system where a mobile client is consuming RESTful Web services using a cloud hosted proxy server. The proxy server also has a cache component. The use of caching in both the mobile client and the proxy server is the extension of the "*Dual Caching Model*" proposed by Liu et al. [29].

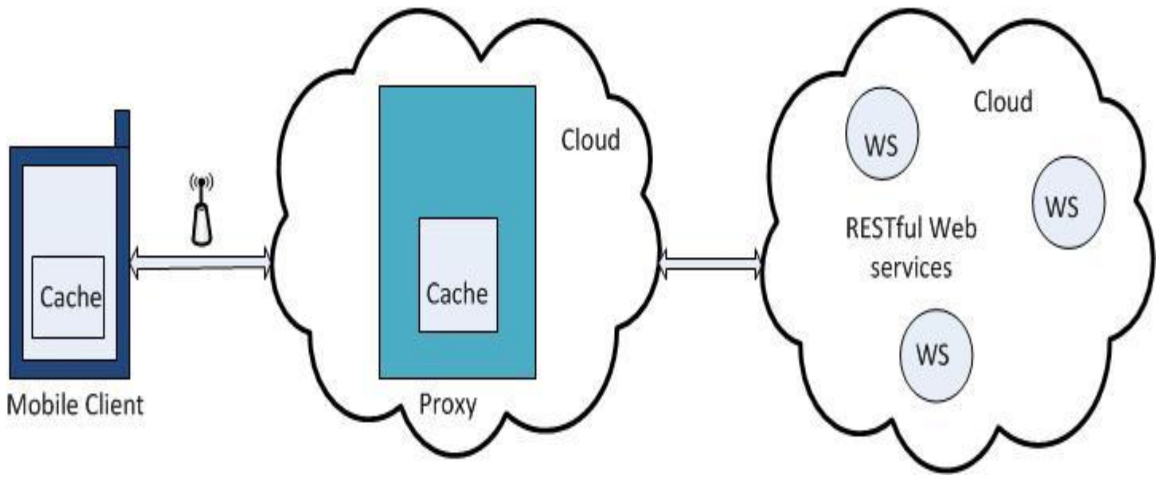


Figure 2-2. Mobile client consuming RESTful WS using caching and a cloud hosted proxy server

In this research, the key issues are:

1. How fast the mobile client and proxy server can efficiently communicate?
2. How to detect and propagate the state changes to the proxy server and mobile client?
3. How to enable the cache to be used in multiple devices of a single user?

To know the current state-of-the-art approaches towards addressing the key issues for the development of the framework will be discussed in the next chapter in the literature review.

CHAPTER 3
LITERATURE REVIEW

This section reviews related research in the following fields: The first part, 3.1 to 3.5 focuses on Web Services, SOA, REST, WADL and REST for mobile clients respectively while the second part discusses two issues which are **network loss** and **latency**. These two issues are typically addressed by caching. Another important aspect within caching is the location of the cache component. It can be on the mobile device, or it can be on intermediary servers such as a proxy server. Therefore, general caching is discussed in section 3.6. Using a proxy server will help with the **disconnecting** and **re-connecting problem** and the feature constraint limitations of the mobile device. Section 3.7 and 3.8 expounds on the idea of proxy servers and the location of the proxy servers respectively.

3.1 Web Services

Web Services (WS) technology is linked to the idea of the Service Oriented Architecture (SOA) [45] [63]. It is a method or technology of interaction between different machines or devices which are accessible through the Internet. According to the World Wide Web Consortium (W3C) [58], a Web service is "a software system designed to support interoperable machine-to-machine interaction over a network." The Web Services Description Language (WSDL) [58] document clearly describes all the specifications for defining a web service in an XML grammar and the capabilities of a web service as well [10] [59]. WS use the Extensible Markup Language (XML) messages for communication over the Internet which is a set of policy defined by the W3C in the XML 1.0 Specification for encoding documents in a machine-readable form [14]. The XML messages or streams "should be validated to ensure that the information can be uniquely understood" while the messages or streams are exchanged for interaction through the network or among applications [28].

The two basic architectures for WS are:

- Service Oriented Architecture, which follows WS* protocol stack, and
- REST

3.2 Service Oriented Architecture (SOA)

The Service Oriented Architecture (SOA) defines everything in terms of services. For a request-response communication, a service consumer and a service provider use messages which are self-containing documents. The SOA based Web service generally follows the Simple Object Access Protocol (SOAP) standards [46] [45]. Pautasso et al. [38] introduces SOA as "Big "Web Services. The OASIS [32] group describes Service Oriented Architecture (SOA) in their reference model as "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains". SOA uses the *find-bind-execute* paradigm which is shown in Figure 3-1 [33].

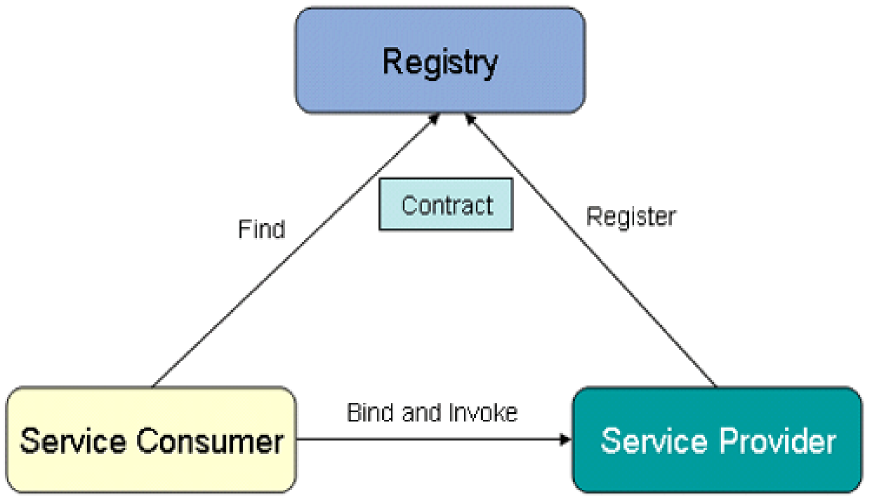


Figure 3-1. Service Oriented Architecture [33]

While describing the Service Oriented Architecture, Box [9] identifies four tenets for service orientation which focus on explicit borders of services, autonomous services, use of schema and policies. The basic unit of communication for SOA is message-oriented, rather than an operation [60]. The key parts heuristic of SOA is that it has a very large protocol which defines a lot of elements and relies on a large specification. And as it relies on messages which consist of XML messages, it produces large amount of traffic [38].

3.3 The Representational State Transfer (REST)

Recently a different approach that has gained popularity for designing Web services is the RESTful Web Service (REST-WS) which was introduced by Fielding [16] in his doctoral dissertation in 2002. REST is a protocol independent architectural design and the acronym stands for "*REepresentational State Transfer"*. The key feature of RESTful WS is that every entity is considered as a resource instead of considering it as a service. It follows a resource-oriented architecture identified by a Uniform Resource Identifier (URI) where every resource has a unique HTTP or similar protocol link. The REST architecture allows simple semantic interfaces to a set of standard operations such as GET, POST, PUT, and DELETE in the HTTP protocol for a client-server interaction [60]. Richardson's Maturity Model [17] identified different types of REST implementation [62] and what are the best and scalable models. The first level talks about POX or Plain Old XML where there will be a "singular service endpoint" and the second level identifies resources where instead of making all the requests to one single port address, there will be multiple resources to be considered. The third level talks about multiple verbs and multiple URIs. This level clearly defines operations that can be done using this standard set of multiple verbs. The highest level, states about "Hypermedia Controls" which means the server drives the clients' states by providing the URI or URL which pre-defines the clients' options and actions as well.

Fielding [16] mentioned the basic principles in his thesis that characterize RESTful systems namely: uniquely identifiable resources, stateful resources, clear operation that modify the state of these resources, and the assumption that the client and server do not know anything about each other except the information they exchange in that conversation. The approach for building REST has been the HTTP protocol. Within this protocol the Unique Identifiers are URIs or URLs. The communication is stateless because the request will not know anything about the client or server. On the other hand, the resources are stateful because they have all the information to represent the message or operation. The CRUD [1] model defines one method for each operation that a resource can do on the server using the HTTP methods [1]. Table 3-1 represents the REST verbs and their corresponding CRUD [1] operations.

Table 3-1. REST verbs and corresponding CRUD operations

REST Verbs	**CRUD Operation**
PUT	Create
GET	Retrieve
POST	Update
DELETE	Delete

Overdick [37] introduces the concept of Resource Oriented Architecture (ROA) as an alternative to the Service Oriented Architecture. This paper is one of the first papers that highlight the fact that REST and SOAP [32] are not very different from one another. He showed that resources are basically services. However, by putting some constrains on unconstrained architecture RESTful architectures can be achieved.

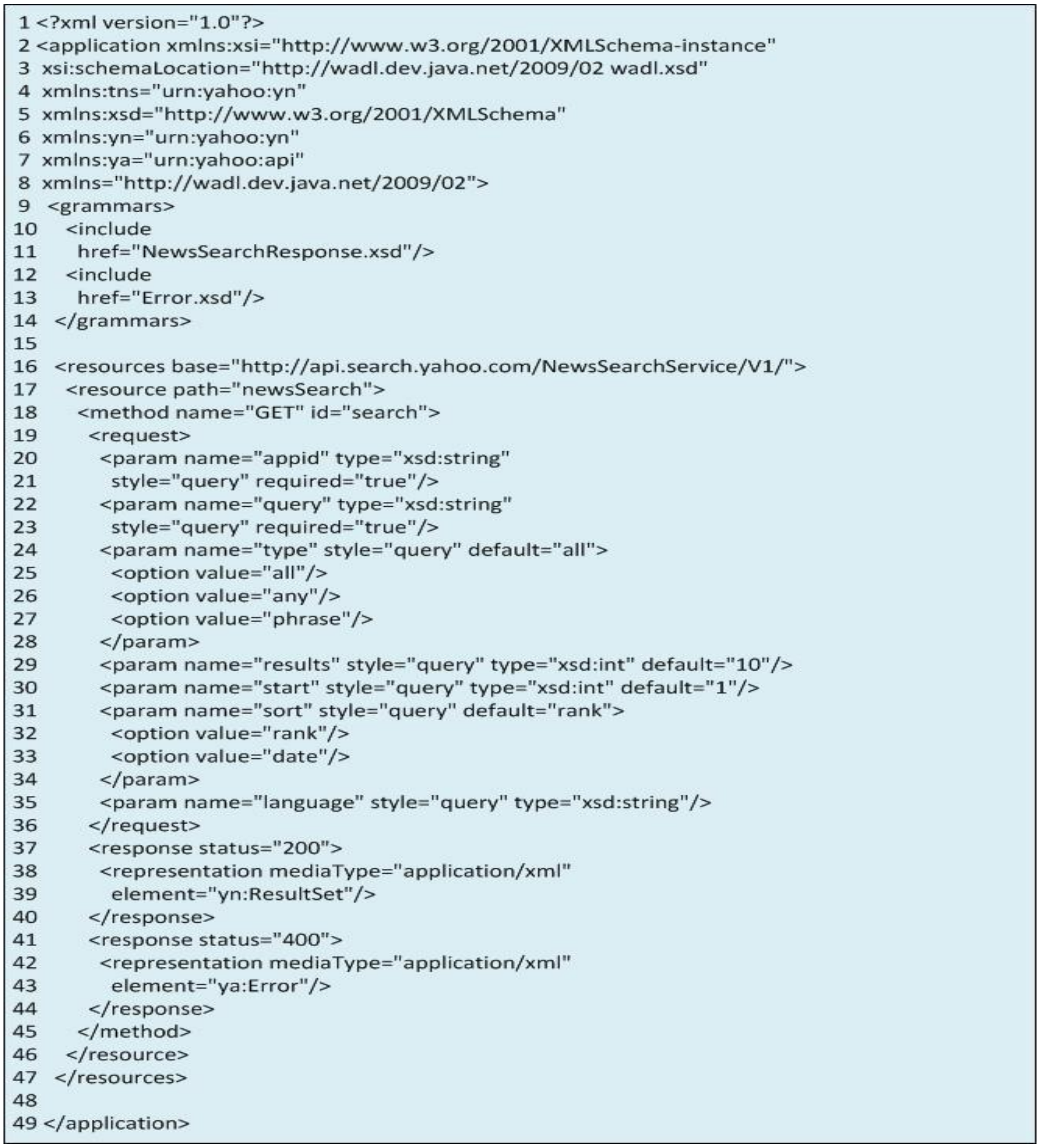
Pautasso et al. [38] compare WS*- services and REST in terms of different implementation levels based on the architectural principles and guidelines. The strength of REST traffic is that it is lightweight, scalable and cacheable. Table 3-2 shows some differences between SOA and ROA.

Table 3-2. Difference between SOA and ROA

SOA	**ROA**
Interaction between services	Interaction between resources
The processes are stateless.	The resources are stateful.
Large protocol	Lightweight [38]
Hard to cache [38]	Cacheable [38]
Semantics are not explicit [2]	Semantics are explicit. [2]
Often tightly coupled [19] [23].	Loosely coupled.[16][38]
Problems with scalability	Highly Scalable [38]

3.4 WADL

The Web Application Description Language (WADL) is an XML based language or specification which is a simple alternative to WSDL (Web Services Description Language) that provides description of HTTP based Web applications which are REST WS that can be read and processed by a machine. It eases the creation of web 2.0 style applications for dynamically configuring services and eliminates the manual writing of the applications [24][57]. According to the W3C [56], “WADL is designed to provide a machine process-able description of HTTP-based Web applications”. Takase et al [49] differentiates WADL and WSDL as a “resource-centric description language” and an “interface-centric description language” respectively. Figure 3-2 is an example of a WADL description for the Yahoo News Search [56] application.



```xml
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
 xmlns:tns="urn:yahoo:yn"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:yn="urn:yahoo:yn"
 xmlns:ya="urn:yahoo:api"
 xmlns="http://wadl.dev.java.net/2009/02">
  <grammars>
    <include
      href="NewsSearchResponse.xsd"/>
    <include
      href="Error.xsd"/>
  </grammars>

  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
      <method name="GET" id="search">
        <request>
          <param name="appid" type="xsd:string"
            style="query" required="true"/>
          <param name="query" type="xsd:string"
            style="query" required="true"/>
          <param name="type" style="query" default="all">
            <option value="all"/>
            <option value="any"/>
            <option value="phrase"/>
          </param>
          <param name="results" style="query" type="xsd:int" default="10"/>
          <param name="start" style="query" type="xsd:int" default="1"/>
          <param name="sort" style="query" default="rank">
            <option value="rank"/>
            <option value="date"/>
          </param>
          <param name="language" style="query" type="xsd:string"/>
        </request>
        <response status="200">
          <representation mediaType="application/xml"
            element="yn:ResultSet"/>
        </response>
        <response status="400">
          <representation mediaType="application/xml"
            element="ya:Error"/>
        </response>
      </method>
    </resource>
  </resources>

</application>
```

Figure 3-2. Example of a WADL description [56]

3.5 REST for Mobile Clients

As a SOA based system needs an XML parser to understand the exchanged message or document which includes additional cost for parsing SOAP [46] requests, Lee et al. [27] propose to implement a ROA-based Web service methodology for Telco. Using low-rest [43] style services they reduce the server side request processing cost because it communicates through HTTP response over the network and the message body contains simple XML or JSON messages. Christensen [11] evaluates the key aspects of using RESTful WS for mobile applications. REST "minimizes the impact of network volatility" and "easy to invoke" as it is URI based and stateless. Another aspect is that REST usually responds with HTTP and thus it is "discrete". REST delivery can be concise and it lends itself to a memory environment which is very constrained [11].

Christensen [11], states RESTful WS are "easier to consume on mobile platforms because the client and server agree on a simple invocation and response protocol. This eliminates the requirement for excessive meta-data based parsing for invocation."

Selonen et al. [43] give an outline for developing a RESTful MR (Mixed Reality) Web Service platform at Nokia Research Center. Mixed reality defines to a combination of real and virtual worlds for "creating environments in which physical and digital objects co-exist. They choose REST and ROA style because they want the system to be secure and scalable. Some additional benefits for choosing REST include "decoupling the clients from the service to support high priority program clients and 3rd party clients in the future, uniform interface to enable evolution of the platform to support new content types over time and aligning with Nokia Services business unit reference stack" [43].

In his research, Stirbu [47] proposes a model to build an adaptive and multi-device application sharing process where the user interface is designed as RESTful architecture. He claims that after implementing it based on RESTful architecture, it is proved that the architecture is "both feasible and effective" [47].

3.6 Caching

Caching is a popular mechanism for improving user experience in distributed client-server communication. Web caching is a way to store previous Web results in order to minimize the communication overhead in a request-response interaction. Falaki et al. [15] investigated the network traffic caused by the smartphone and found that more than half of the traffic is contributed by Web browsing. To deal with network traffic, connectivity problem and latency, caching becomes vital for mobile clients. Cached data for a particular client request can be stored on the client side for any equivalent future requests. Client cache improves performance, advances system scalability and reduces latency, especially because of partial reduction of few interactions.

Liu et al. [29] proposed a dual caching model where caches are put on nomadic clients and the server. They showed that it is possible to have two caches where the client cache shields the mobile client from the interconnectivity problem and the wired side cache shields the loss of connectivity or spotty connection problem. With the proxy side cache, pre-fetching [29] and caching can be used which shields the client from the complexity that arrives from the client disconnection and re-connection problem.

To maintain consistency of data in mobile distributed systems, Wang et al. [54] [55] evaluated a scheme called “scalable asynchronous cache consistency scheme (SACCS)”. They proposed the scheme to maintain cache consistency between the mobile support station (MSS) and mobile user’s (MU) caches.

Ren et al. [41] proposed to use a “semantic caching scheme” while accessing Location Dependent Data (LDD) or application in mobile environments.

Frangiadakis et al. [18] did research on investigating the existing caching strategies for highly mobile environments and states that “efficiently answering multiple queries” or “reception of the data” is preferred than “small delays in delivery”. They provide analysis of push, pull and hybrid push-pull ratio for efficiently receiving data in a mobile environment.

To reduce energy consumption and latency in mobile applications, Shen et al. [22] proposed the “Greedy Dual Least Utility mechanism”, a novel caching method with a cache replacement algorithm and a passive pre-fetching algorithm. However, caching becomes critical for mobile service clients due to connectivity problem and limited bandwidth.

The third feature of REST introduced by Fielding [16] in his doctoral dissertation is “client-cache-stateless-server”. REST is absolutely vital for mobile clients because here caching and proxies can be used as notion of the state is clear. The semantics are well known [2] and changes in states are clear which is very important for caching.

3.7 Proxy Servers

A proxy server is a “server that acts as an intermediary for requests from clients seeking resources from other servers” [39]. Proxy server serves the response requested by a client by connecting with the original server on behalf of the client. To do so the proxy server checks the requests for validation through filtering before connecting to the appropriate server for serving the service. On the other hand, a proxy server can cache responses which allows it to serve clients’ request for the same data without contacting the original server [39]. Figure 3-3 illustrates how a proxy server works.

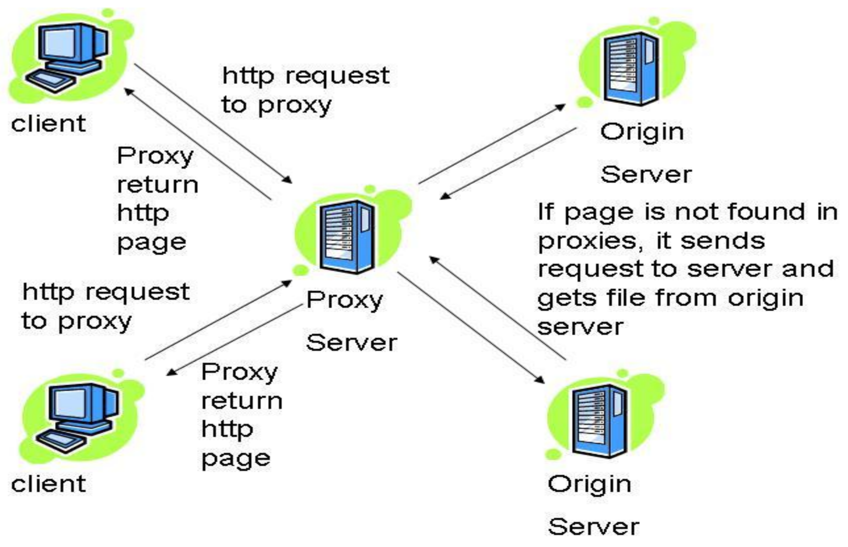


Figure 3-3. Working of Proxy Server [64]

Endler et al. [12] in their survey based report investigate various proxy based system approaches, main categories of tasks allocated on them and frameworks for the development and deployment of proxy based systems for mobile computing. They argue that proxy based systems are the "best solutions" for mobile applications. They propose the need of using "dynamic proxy configuration, which allows shaping the proxy's functionality according to dynamic demand by the clients, server load, or the current mobile network conditions" [12]. They also discuss the necessity of proxy side caching and different approaches of caching in mobile computing.

3.8 Cloud Computing

Schmidt introduced the term "cloud computing" in a conversation with Sullivan on Search Engine Strategies Conference in 2006 by saying "data services and architecture should be on servers. We call it cloud computing – they should be in a 'cloud' somewhere" [42]. According to Armbrust et al. [6], today computing is accessible in a way where users can access the services based on their requirements like utility services such as electricity, gas, water, and telephone.

Cloud computing is a new paradigm of distributed computing that allows software and data accessing, and storage services without end-user's knowledge of where the servers are located, how the services are delivered and maintained [44]. Armstrong et al. [6] and Barnatt [7] state cloud computing as "the applications delivered as services over the Internet". Figure 3-4 illustrates a Cloud computing conceptual diagram.

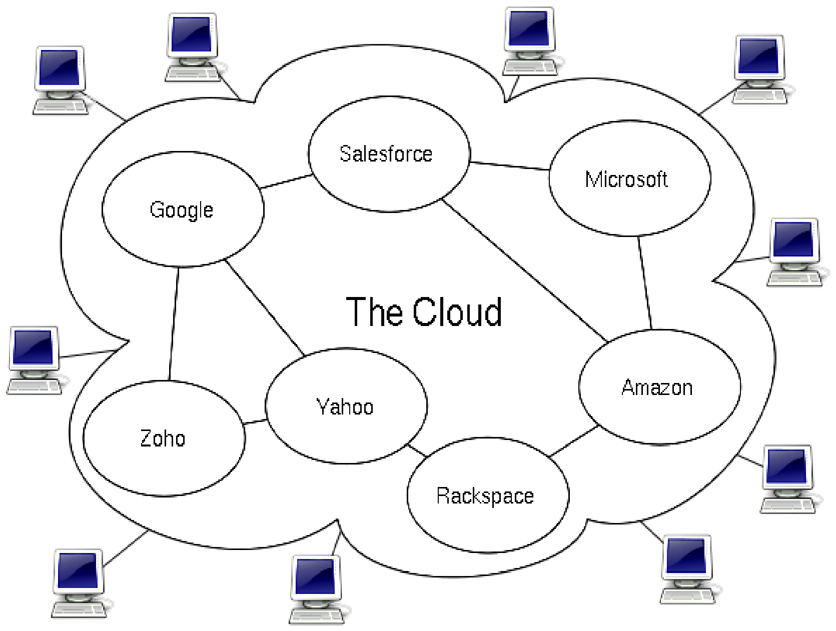


Figure 3-4. Cloud computing conceptual diagram [44]

According to "Evans Data" [50], after conducting a survey with over 400 software developers to determine their perceptions of leading cloud space vendors and providers, EC2, GAE, IBM's cloud and Microsoft's Azure are the most popular and top list companies. Amazon Elastic Cloud Computing (EC2) provides a Virtual Environment where users can choose different Operating Systems and hardware architectures to run on their virtual machines [4]. Vaquero et al. [52] name "scalability, pay-per-use utility model and virtualization" as the key features while defining cloud computing. They classify three major scenarios in cloud computing based on the type of provided capability namely: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

According to them, there are 3 major styles depending on resource abstraction technologies for cloud computing:

1. **Infrastructure as a Service (IaaS)** refers to a service that exposes the hardware resources to users. Amazon EC2 is a successful IaaS implementation in the market [52].

2. **Platform as a Service (PaaS)** provides computational resources as high level application platforms. Google App Engine (GAE) is an example of PaaS [52].

3. **Software as a Service (SaaS)** focuses on exposing software functions as services (i.e. WS) [52].

Qian et al. [40] study the underlying idea, history and status, advantages and risks, value chain and standardization of cloud computing. They define five key technical features of cloud computing: large scale computing resources, high scalability & elastic, shared resource pool, dynamic resource scheduling and, general purpose [40]. Armbrust et al. [6] believes that with cloud computing, some existing applications will become "more compelling and contribute further to its momentum" and "Mobile interactive applications" is one of them.

3.9 Summary

Mobile devices are resource constrained clients. To deal with network traffic, limited resources, connectivity problem and latency, caching becomes vital for mobile clients [15]. The "*Dual Caching Model*" which introduces caching for both client and server sides, can be the solution of the generic problems which arise while consuming WS in mobile clients [29]. Proxy servers can be used to shield clients from the emerging complexities while connecting with the server for resources [12]. Some strength features of REST traffic is that it is lightweight, scalable, cacheable [38] and loosely coupled [16] [38]. The use of REST becomes very important for mobile clients since caching and proxies can be used as the notion of state is clear. Hosting the servers in a cloud computing environment can enhance the scalability and reliability of mobile clients [36]. And also with cloud computing, mobile applications can be made more credible [6]. Table 3.3 shows the solutions found from the previous researches.

Table 3-3. Solutions of the 4 problems found from reviewed research

Issues	**Findings**
Network loss/ Temporary network loss	In order to deal with network loss, the client must have a cache [29], [15], [18], [12]. In case we are dealing with temporary outrages, it is also good to have a proxy on the wired side to allow the client to re-connect and continue with the session [39], [12].
Latency	Caching in client side and also in proxy side is the reason for reducing latency or speed up the information access over the network. This rages the question of how to do pre-fetching to reduce latency [29], [15], [18], [12].
Limited Resources	To deal with resource constrains, use of a proxy server and also hosting the proxy server in the cloud is a good solution. To deal with limited resource problem, pre and post processing in a proxy closer to the network resources can be one possible solution [6], [7], [36], [12].
Reduced Bandwidth	This is again primarily the goal of using client side caching and middleware for example a proxy server which sends the updates. We get the idea that we can deal with this problem with the use of lightweight protocol and push through the proxy server [29], [15], [18], [39], [12].

In summary, the reviewed research indicates:

- A cache is needed on the mobile client side [29] [15] [18] [12].

- A middleware or proxy server is needed which is going to have a cache and also going to update the mobile client side cache as well as to deal with the consistency issue [29] [12].
- Hosting the servers in Cloud can enhance the scalability and reliability of mobile clients and mobile applications can be made more credible [6] [36].
- As REST traffic is lightweight and scalable, the use of REST becomes very important for mobile clients since caching and proxies can be used as the notion of state is clear [16] [38].

However, there are still **open questions** regarding these findings that remain, namely:

- How to detect the state changes and how to push state changes on the mobile client?
- How to build a portable cache which can be used for multiple devices?
- How to build and organize the caches so that they can be used for multiple devices of a user?
- How to deal with the heterogeneous Web resources as we have so many different resources and everyone has a different policy, different mechanisms for propagating its updates?
- How to implement the proxy server and the RESTful Web server on cloud platforms?
- How to link these resource constrain mobile devices wirelessly to different RESTful WS?

While describing these issues, everyone debated about the sub problems regarding these questions. However, no one has actually built a complete architecture. Even if they did this, they did not test it in realistic settings.

CHAPTER 4
ARCHITECTURE

Within this research, the main emphasis is on RESTful Web Services (REST-WS) that follow the HTTP protocol. Every WS is to be considered as a different resource/object with different policies. The mobile device will maintain multiple caches which are key/value pair storages for each individual resource/object. Figure 4.1 illustrates the proposed architecture.

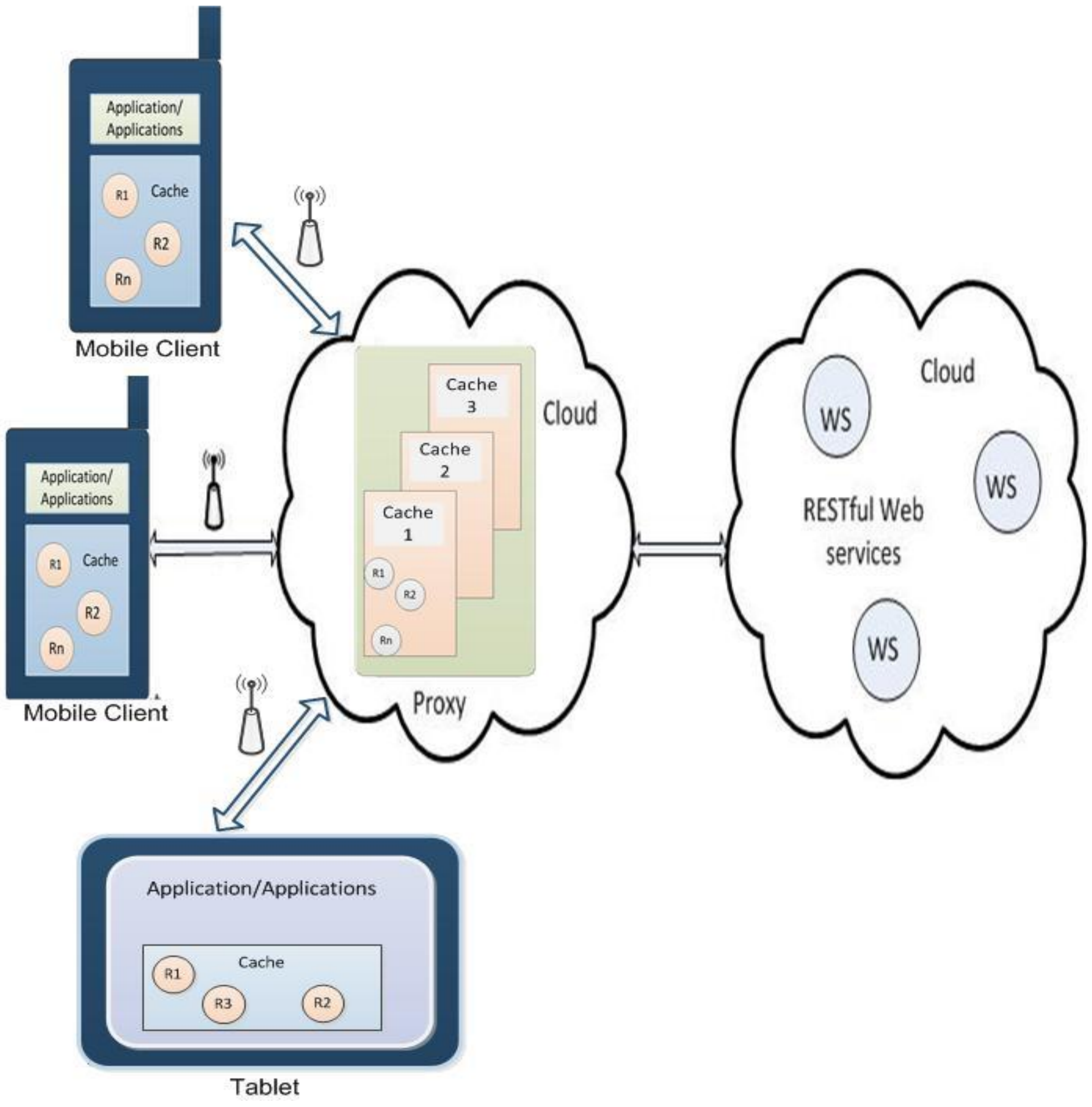


Figure 4-1. The proposed architectural design

In this work, the goal is to investigate the approaches for caching and the use of a proxy server to overcome some of the challenges like network loss, bandwidth consumption, resource constrains and latency. The architecture (Figure 4-1) consists of three parts, the mobile clients, the proxy server and the RESTful WS hosted by REST Web server. Both the proxy server and WS are in the Cloud. The caches will be on the mobile device itself and also in the proxy server which will sit between the mobile device and the REST Web server. The mobile devices have individual caches consisting of every mirrored resource. The proxy server also has caches for every single resource. The use of caching in both the client side and the server side is the extension of the "*Dual Caching Model*" proposed by Liu et al. [29].

4.1 Mobile Client

The mobile client hosts one or multiple applications. There will be caches in the mobile device itself which will have "mirrored resources" that are copy of RESTful resources from the REST Web server. Whenever a state change or update happens in the resources, the client caches will get them through the proxy server. Each resource can have their own policies such as different caching and pre-fetching strategies. Figure 4-2 shows the architecture of the mobile client.

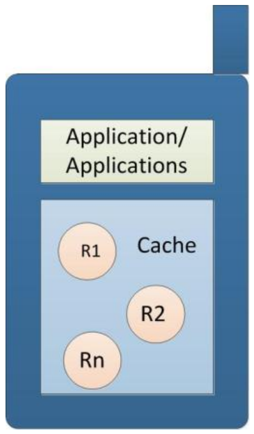


Figure 4-2. Architecture of Mobile Client

4.2 Proxy Server

The proxy server has multiple sub proxies in it and resources like R1, R2, and R3 for mobile client1; resources R1, R2 and R3 for mobile client2 and so on as a user can have multiple mobile devices. There will be caches for every resource. They can be shared in multiple devices or not. In a generic proxy server, all users share the same cache making the process complex as they have to maintain information about each user. However, in the proposed approach, every user can now have its own set of information. This proxy server hides all the issues in the wired world from the mobile client and knows how to interpret messages from these resources. The proxy server will have multiple caches for every single mobile client of a user. Figure 4-3 shows the internal architecture of the proxy server.

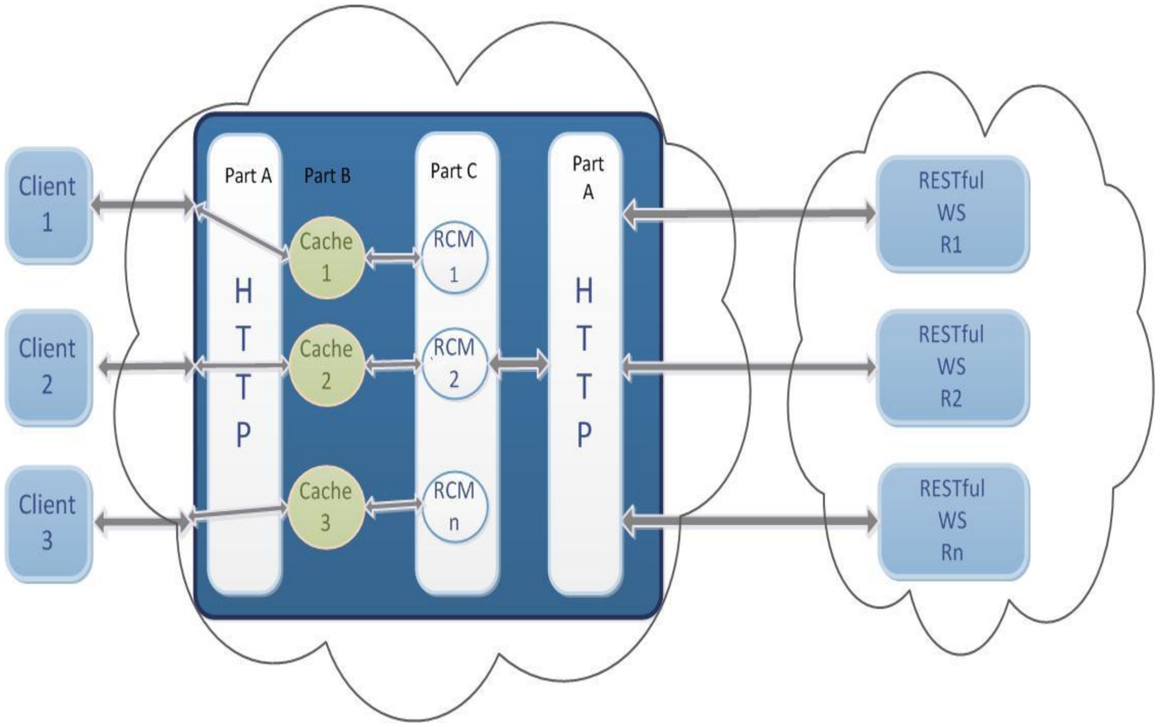


Figure 4-3. Architecture of the proxy server

The proxy server consists of:

- Part A: HTTP Server components that communicate with the mobile clients and the RESTful WS.
- Part B: Data store which are key/value pair storage caches where for every resource we will have caches.
- Part C: Controller which works as Resource Cache Manager (RCM). RCM executes the requests and gets the result back. Every resource will have its own RCM. The Web server will connect with them and notify the state change to RCM. If there is one user with two devices, for each resource there will be one resource cache. The more the user shares resource cache, it is easier for the proxy server to operate. When there is a state change in one of the resources, they get updated in all of them. Figure 4-4 shows the internal architecture of the proxy server for multiple devices.

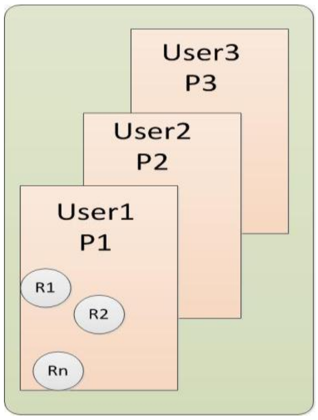


Figure 4-4. Cache representation on the Proxy Server for multiple devices

4.3 Proxy Object

In the proposed architecture, the proxy server contains resources which are proxy objects. The proxy objects know how often to query for services, when and where to send updates. The proxy object knows how to interpret messages of the resources and the meaning of a new state of the resource in the server. It encapsulates details of state changes in the server and the interpretation. Either push method or pull method is applied; these proxy objects contain all the information. These proxy objects also have the goal to later communicate to the client side proxies and can have their own proprietary protocols to push data to the client. So the proxy object has two different sides: the server side and the client side. The server side will know about resources (e.g. resource R1 which is shown in Figure 4-5) in the actual service and interpret its actions, how to send messages and how to understand if a state change is happened. The client side which after detecting the state change, sends it to the client. It will also know how to send information to the other clients. Figure 4-5 illustrates the two goals of a resource R1 in a proxy server.

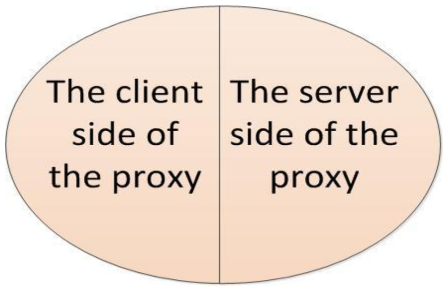


Figure 4-5. Two goals of Object R1 in the proxy server

Question 1: How will the client cache and proxy server cache communicate?

Suppose the mobile client requests for resource R1. The client application will first go to the client cache to see if the resource is already in the cache or not. If it is already there, it will get the data from the cache and send it to the application. If it is not in the cache, the application will connect with the proxy server. The communication between the client cache and proxy server cache will be over HTTP. The proxy server will get the data from the proxy server cache if it’s there. Otherwise the proxy server will get the data from the RESR Web server, parse it and form a new service results in JSON format, stores the result in the proxy server cache and send back the optimized result to the mobile client in JSON format. Figure 4-6 illustrates the communication between the client and proxy server.

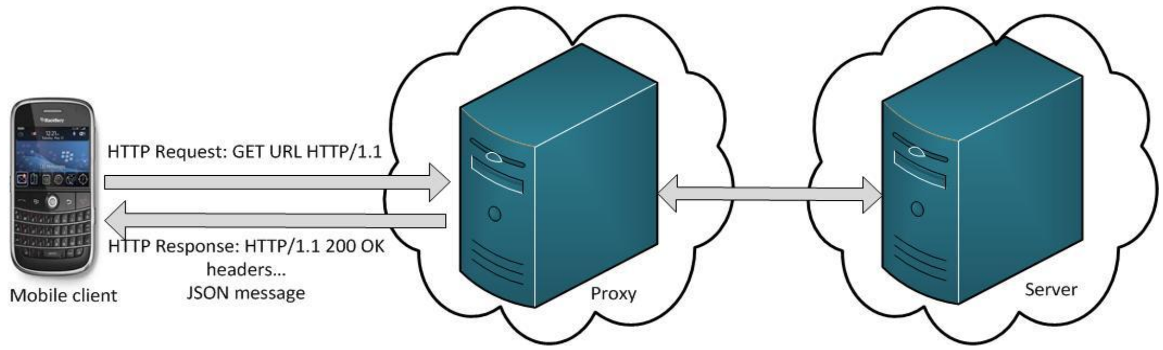

HTTP Request: GET URL HTTP/1.1
HTTP Response: HTTP/1.1 200 OK
headers...
JSON message
Mobile client
Proxy
Server

Figure 4-6. Communication between the client and the proxy server

Question 2: How to propagate the state changes?

The proxy server is responsible for consuming the WS, getting the state changes, parsing, caching and delivering the service result to the mobile client. To get the state changes the proxy server sends a HEAD request to the REST server whenever there is a GET request coming from the mobile client. If the ETAG [51] value, which is a unique entity of the Web resource, has changed, that means the state is changed. To get the state changes, there are several possibilities and I am going to evaluate them.

1. **Proxy server A:** The proxy server checks on the resources by sending HTTP HEAD requests to the REST Web server when the mobile client sends GET request for a particular resource, the proxy server grabs it from the REST Web server if it is not in the proxy server cache, stores it and sends it to the client which includes other updates from the REST Web server as well. This method is also called “Gossiping” [3]. Figure 4-7 illustrates the Gossiping method of state change.

2. **Proxy server B:** The proxy server checks on the particular resource by sending HTTP HEAD request to the REST Web server when the mobile client sends GET request for a particular resource, the proxy server grabs it from the REST Web server if it is not in the proxy server cache, stores it and sends it to the mobile client from the proxy server. Figure 4-7 illustrates the system.

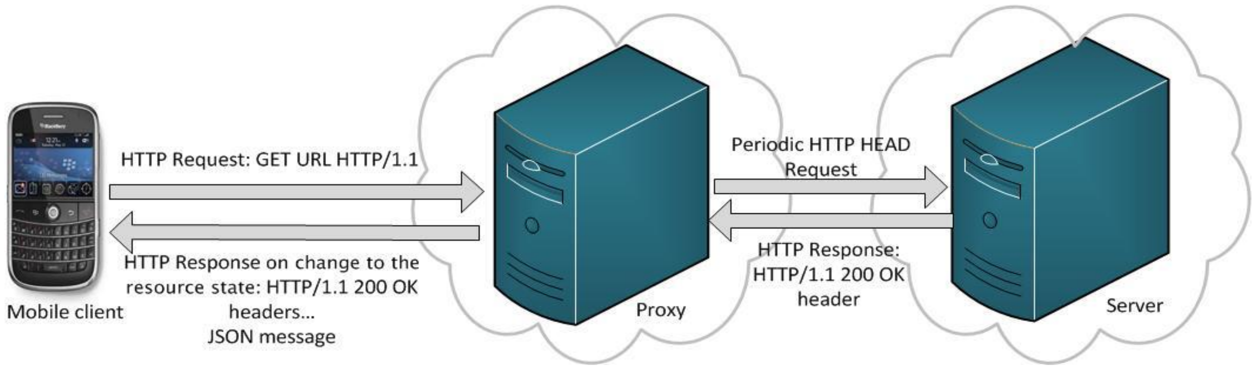

HTTP Request: GET URL HTTP/1.1
HTTP Response on change to the resource state: HTTP/1.1 200 OK headers... JSON message
Mobile client
Proxy
Periodic HTTP HEAD Request
HTTP Response: HTTP/1.1 200 OK header
Server

Figure 4-7. State change propagation

Question 3: How to build a portable cache which can be used for multiple devices of a user?

Suppose a user has N mobile clients. The proxy server will maintain N caches for every single mobile client for a user. If mobile client1 requests for resource R1 and it connects to the proxy server, it can get the response back and can create its own cache. Or it can select an existing cache from the proxy server. The end user can delete its existing cache as well. For authentication, security tokens can be sent with the GET request from the client. As every resource has its own cache, a mobile client can access with security tokens for a particular resource. The workflow of the implementation that takes place when the user has a browser based application running in the mobile client and issues a request for a service is described in Figure 4-8.

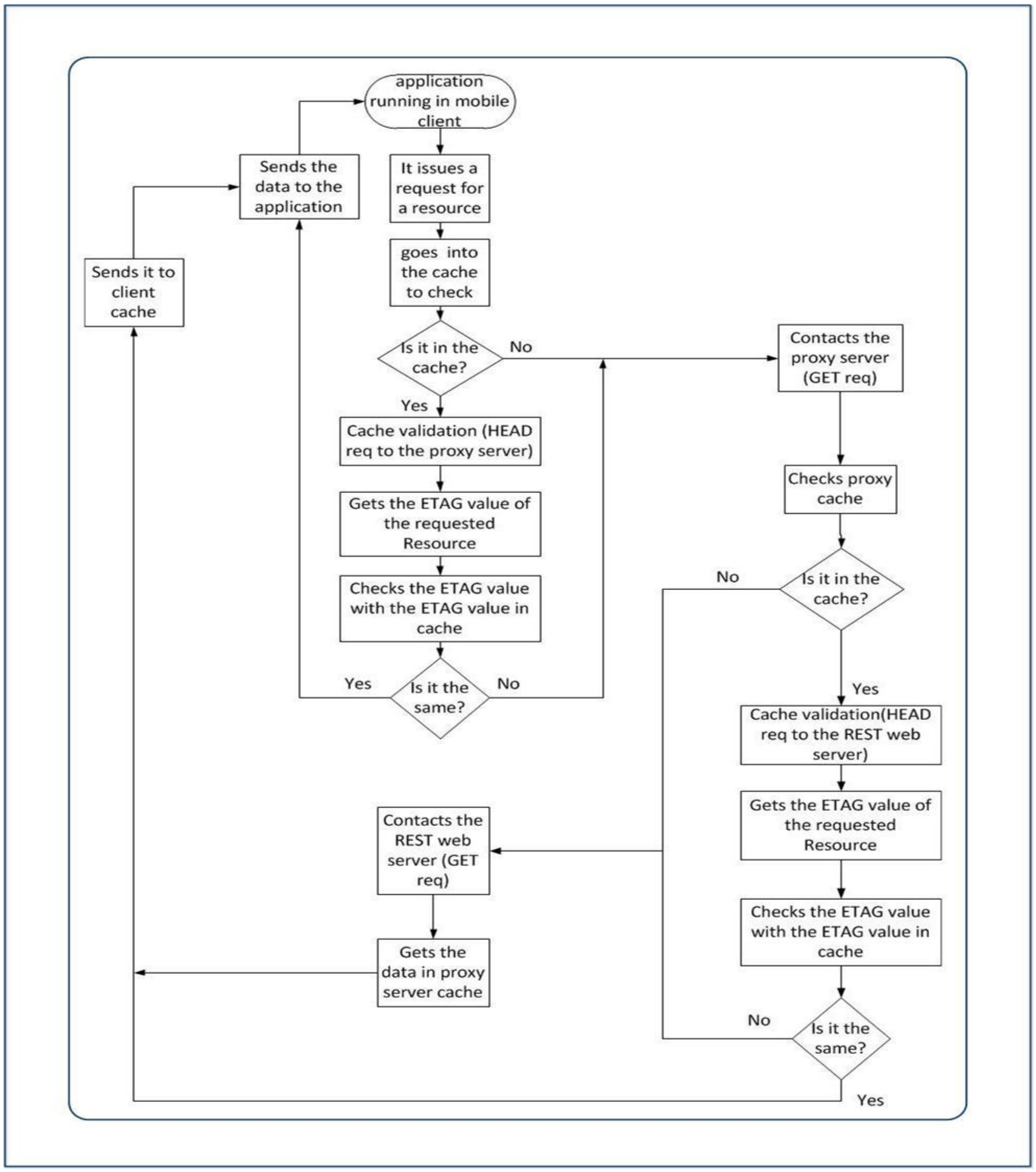

application running in mobile client
It issues a request for a resource
goes into the cache to check
Is it in the cache?
No
Yes
Cache validation (HEAD req to the proxy server)
Gets the ETAG value of the requested Resource
Checks the ETAG value with the ETAG value in cache
Is it the same?
Yes
No
Sends the data to the application
Sends it to client cache
Contacts the proxy server (GET req)
Checks proxy cache
Is it in the cache?
No
Yes
Cache validation(HEAD req to the REST web server)
Gets the ETAG value of the requested Resource
Checks the ETAG value with the ETAG value in cache
Is it the same?
No
Yes
Contacts the REST web server (GET req)
Gets the data in proxy server cache

Figure 4-8. Workflow diagram of the system

CHAPTER 5
IMPLEMENTATION

To evaluate the use of a cloud hosted proxy server as a means for supporting smartphones, I implemented the architecture shown in figure 4-1 (Chapter 4) and built a prototype application called “GradeView App”. The architecture is implemented using BlackBerry’s WebWorks framework and Android’s WebView framework as the client application and two servers that are identical to EC2 (Amazon) as a cloud provider. Using the application, a student will be able to access his grading records for different courses using his mobile device that are hosted on a REST Web server. To simplify the evaluation the BlackBerry 9550 Simulator and Android emulator are used as the mobile clients. The application is also deployed on a BlackBerry Playbook tablet device for testing.

The client application is BlackBerry Web Work application and Android WebView application which is the embedded browser framework. Embedded browser framework is a device or platform independent Web based framework that allows Web Services to be deployed as resident applications on the mobile device. Figure 5- 1 presents the architecture of the WebWorks/ WebView.

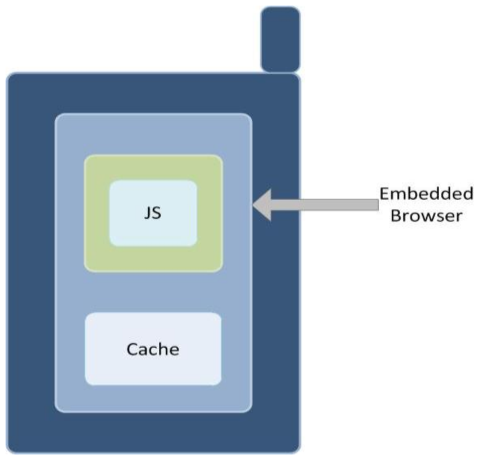
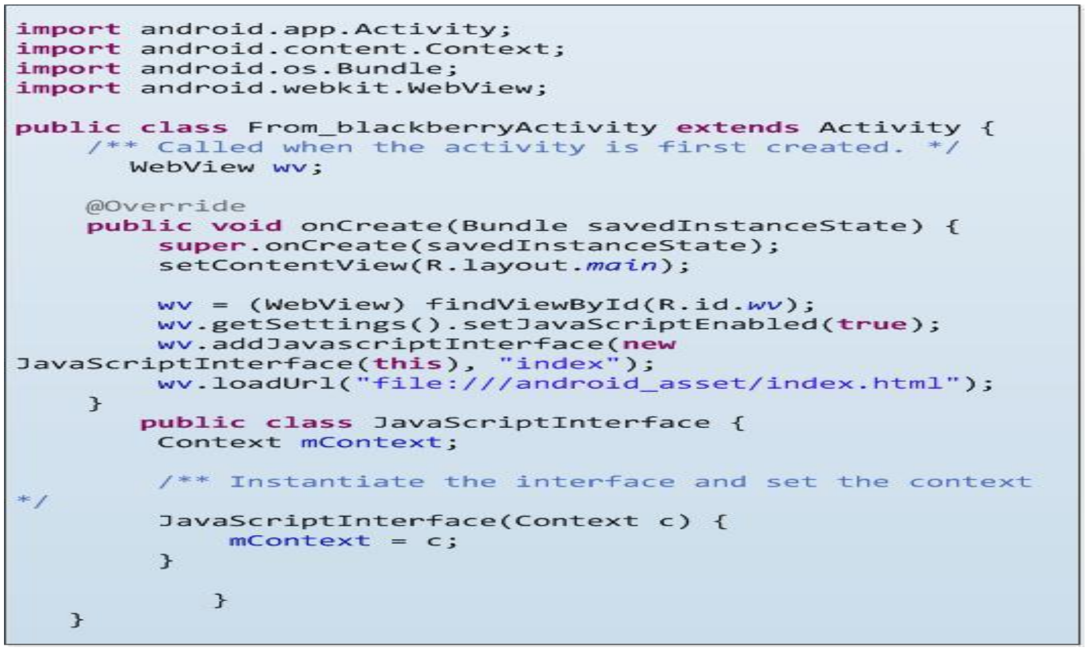


Figure 5-1. WebWorks/ WebView Architecture

Using WebWorks/WebView, the bulk of the client application is implemented using JavaScript. The JavaScript has access to the cache component that is developed in Java and is accessible via the JavaScript-Java bridge. The cache component intercepts all the HTTP requests and decides based on connectivity and available cache data, how to process them. If the cache component does not have the requested data, then it connects to the proxy server for the response. A proxy server is used as the intermediary between the BlackBerry/ Android client and the RESTful WS which are hosted on a REST Web server. The proxy server is developed in Erlang [13] to ensure maximum scalability and high dependability when serving mobile consumers. For the RESTful WS for hosting the grading records, a database server is used (CouchBD [5] server is used which is a document oriented Non-Relational Database Management Server). Figure 5-2 shows the code snippet for the Android WebView to create a simple activity that enables JavaScript interface and Figure 5-3 shows the UI of the application in the Android emulator.



```
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.webkit.WebView;

public class From_blackberryActivity extends Activity {
     /** Called when the activity is first created. */
        WebView wv;

     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);

          wv = (WebView) findViewById(R.id.wv);
          wv.getSettings().setJavaScriptEnabled(true);
          wv.addJavascriptInterface(new
JavaScriptInterface(this), "index");
          wv.loadUrl("file:///android_asset/index.html");
     }
         public class JavaScriptInterface {
          Context mContext;

          /** Instantiate the interface and set the context
*/
          JavaScriptInterface(Context c) {
              mContext = c;
          }

              }
   }
```

Figure 5-2. Code snippet for Android WebView

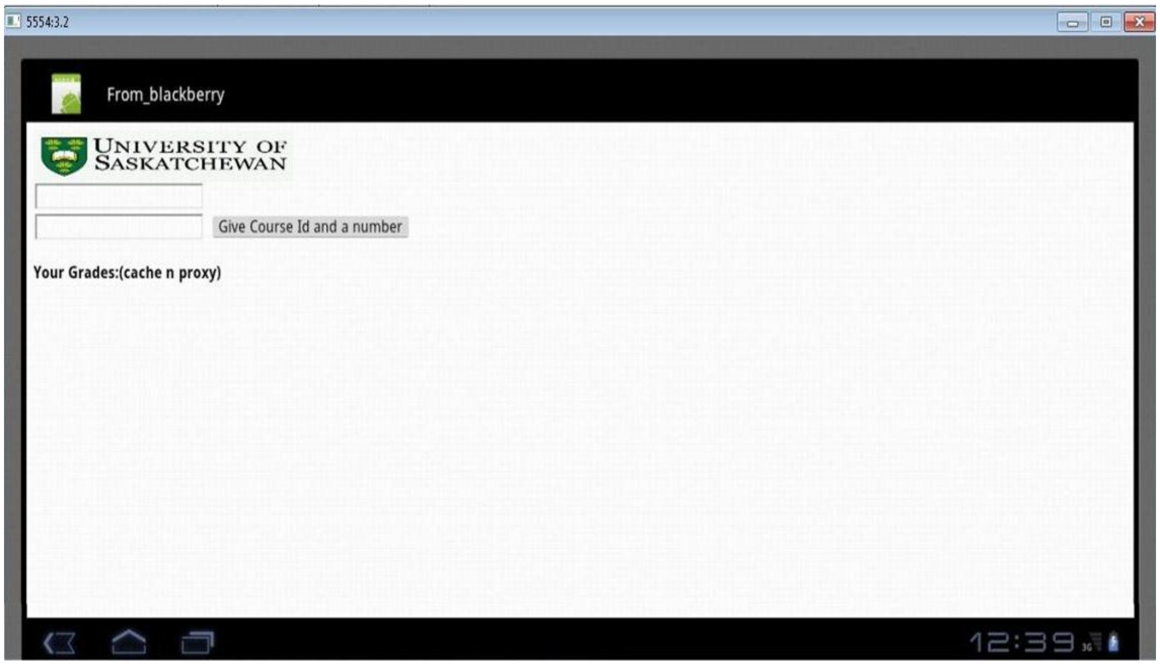
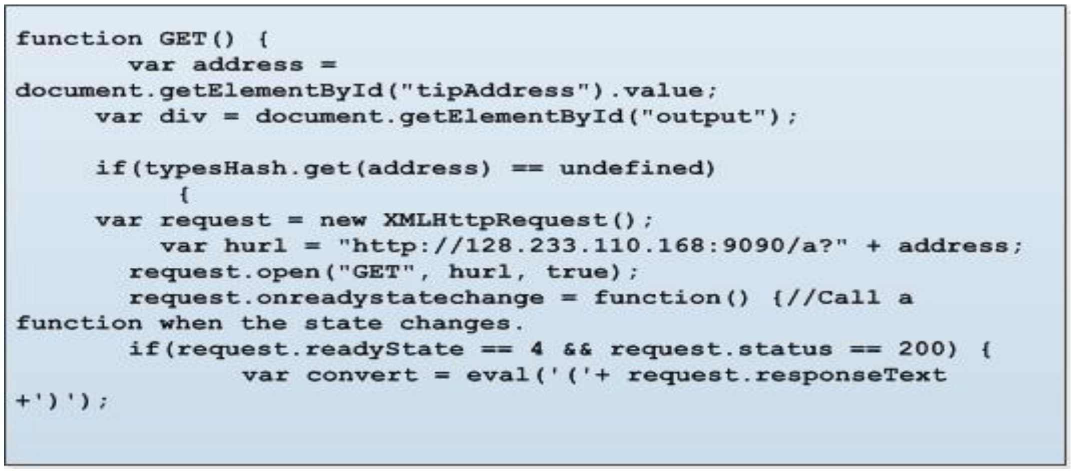

5554:3.2
From_blackberry
University of Saskatchewan
Give Course Id and a number
Your Grades:(cache n proxy)
12:39

Figure 5-3. The UI of the application using WebView in Android Emulator

Figure 5-4 shows the code snippet for the BlackBerry WebWorks to send a HTTP GET request which is created using the XMLHttpRequest() class in JavaScript to send asynchronous requests to the proxy server. Figure 5-5(a) shows the UI of the application in the BlackBerry mobile simulator and Figure 5-5(b) shows the UI of the application in the BlackBerry Playbook tablet device.



```
function GET() {
       var address =
document.getElementById("tipAddress").value;
     var div = document.getElementById("output");

     if(typesHash.get(address) == undefined)
          {
     var request = new XMLHttpRequest();
         var hurl = "http://128.233.110.168:9090/a?" + address;
       request.open("GET", hurl, true);
       request.onreadystatechange = function() {//Call a
function when the state changes.
       if(request.readyState == 4 && request.status == 200) {
              var convert = eval('('+ request.responseText
+')');
```

Figure 5-4. Code snippet for BlackBerry WebWorks

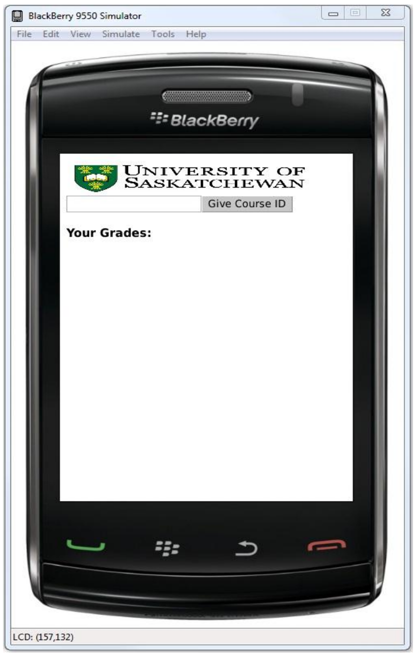

BlackBerry 9550 Simulator
File Edit View Simulate Tools Help
BlackBerry
UNIVERSITY OF SASKATCHEWAN
Give Course ID
Your Grades:
LCD: (157,132)

Figure 5-5(a). The UI of the application using WebWorks in BlackBerry mobile simulator

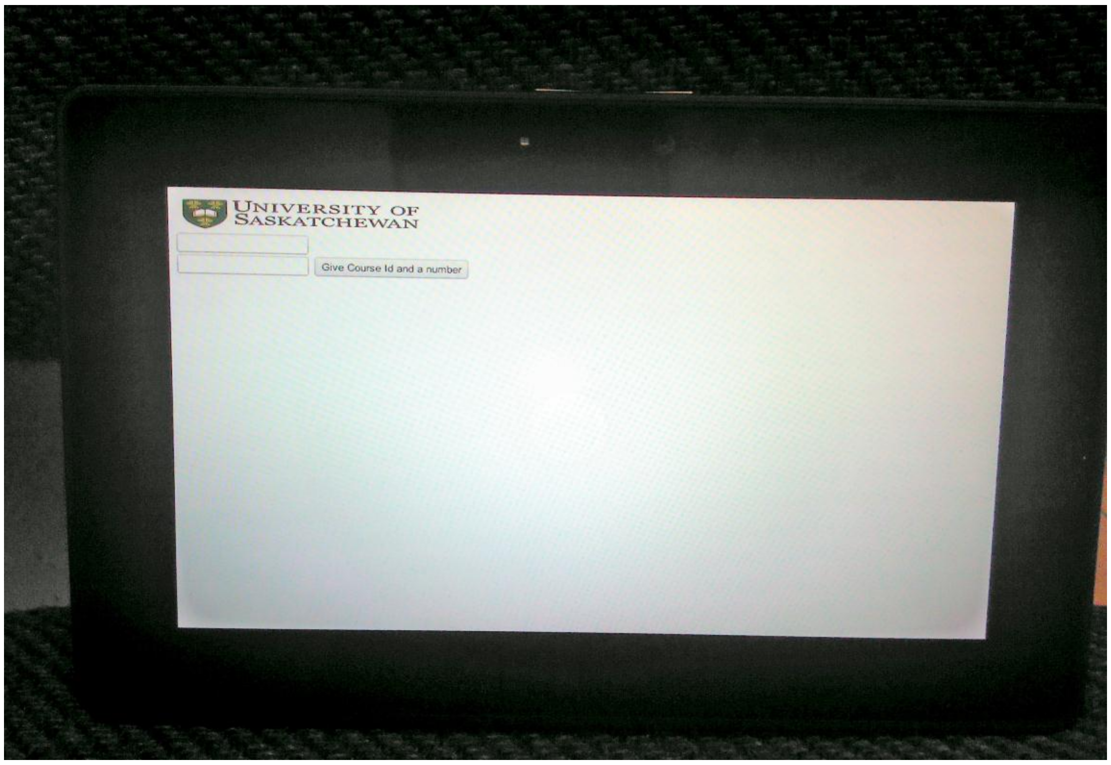


Figure 5-5(b). The UI of the application using WebWorks in BlackBerry Playbook tablet device

5.1 Cache Organization of the client

The four different cases for the cache organization of the client are as follows:

Case 1: When the resource is found in the client cache and there is no update in any of the resources: When a student wants to know his grade for a particular course, he gives the course id in the GradeView application in his mobile device. The GradeView application, which is the client application, issues a request for the requested resource. It first goes to the client cache to check if the resource is there or not. If the resource is found in the client cache, it issues a GET/HEAD request to the proxy server to get the "ETAG" [51] (which is a unique entity of a Web resource to determine whether the local copy of the cache has an updated version or an old version of the Web resource) of the resource to check if there is any state change happened in the resource by matching the recent ETAG value with previous ETAG value which is already stored in the client cache. The proxy server checks if the resource is in the proxy cache or not. The proxy server also issues a GET/HEAD request to the REST Web server to check if there is any state change happened to any of the resources or not. If the resource is not found in the proxy cache it fetches the resource from the REST Web server. If there is no state change, the ETAG is returned to the client cache. As there is no change in the "ETAG" and it matches with the previous value which is in the cache of the client application, the response will be retuned directly to the UI of the client application from the client cache **(Figure 5-6(c) Condition 3)**.

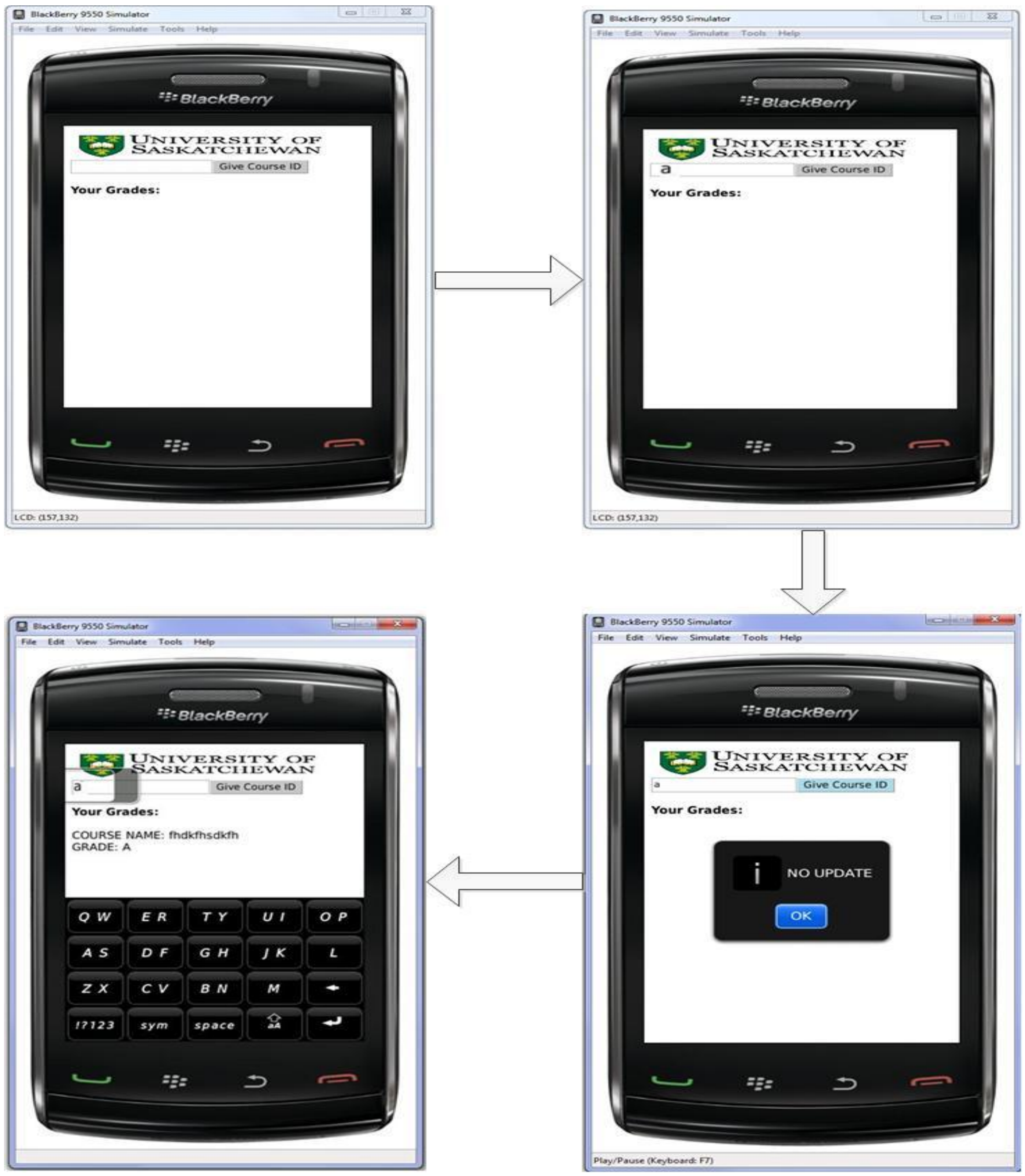
Case 2: When the resource is found in the client cache and there is update in any of the resources: If the ETAG value is different from the previous value, then the GradaView application contacts the proxy server with a HTTP GET request. The proxy server contacts the REST Web server for the resource and pulls the response of the resource and the updates of other resource as well from the REST Web Server. The proxy also keeps a copy of it in the proxy cache and pushes the response to client's cache. Then the client cache keeps a copy of the resource in the storage with the updates for other resources and provides the response to the UI of the application **(Figure 5-6(d) Condition 4)**.

Case 3: When the resource is not in the client cache and there is no update in any of the resources: If the resource is not in the client cache, it contacts the proxy server. The proxy server checks if the resource is in proxy cache or not. If the resource is not found in the proxy cache it fetches the resource from the REST Web server. The proxy also issues a GET/HEAD request to the REST Web server to check if there is any state change or update happened to any of the resources or not. If there is no state change to any other resources, the response is returned to the proxy cache which has the requested resource only, sends it to the client cache and provides the response to the UI of the GradeView application **(Figure 5-6(a) Condition 1)**.

Case 4: When the resource is not in the client cache and there is update in any of the resources: If there is any state change happened in any of the resources in the REST Web Server, the proxy server contacts the REST Web server for the resources, and keep a copy of them in the proxy cache and pushes the response which includes the response for the asked resource and the responses for other updated resources to client's cache. Then the client application keeps a copy of the resources in the client cache with the updates for other resources and provides the response to the UI of the application **(Figure 5-6(b) Condition 2)**.

The four different cases of cache organization process is showed by Fig 5-6(a), Fig 5-6(b), Fig 5-6(c) and Fig 5-6(d).

***Condition 1.* When the resource (requested grade) is not in the client cache and there is no update in any of the resources (grades):**


Your Grades:

a

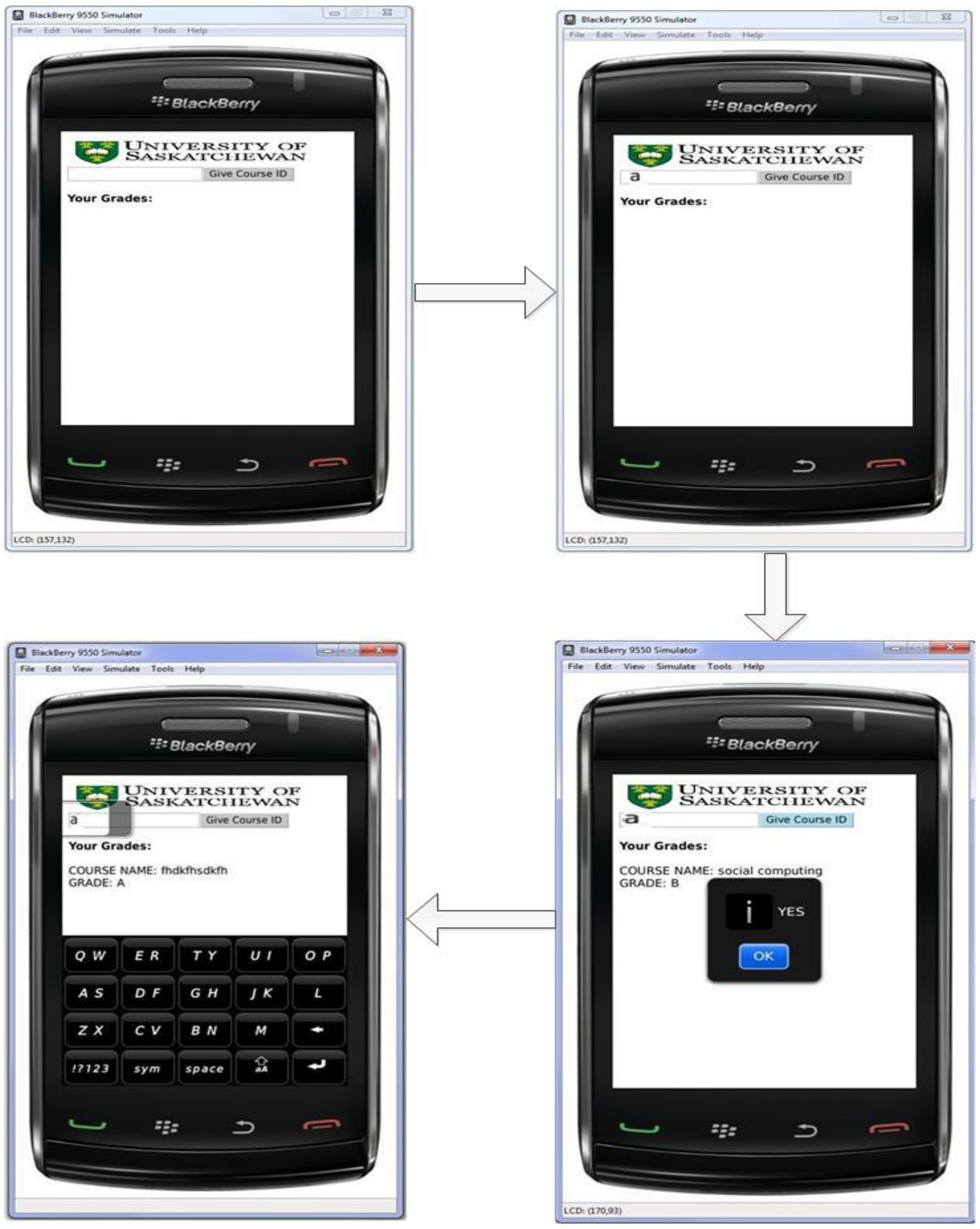
NO UPDATE

COURSE NAME: fhdkfhsdkfh
GRADE: A

BlackBerry 9550 Simulator
File Edit View Simulate Tools Help
BlackBerry
UNIVERSITY OF SASKATCHEWAN
Give Course ID
OK
LCD: (157,132)
Play/Pause (Keyboard: F7)

Figure 5-6(a). Condition 1

***Condition 2.* When the resource (requested grade) is not in the client cache and there is update in any of the resources (grades):**

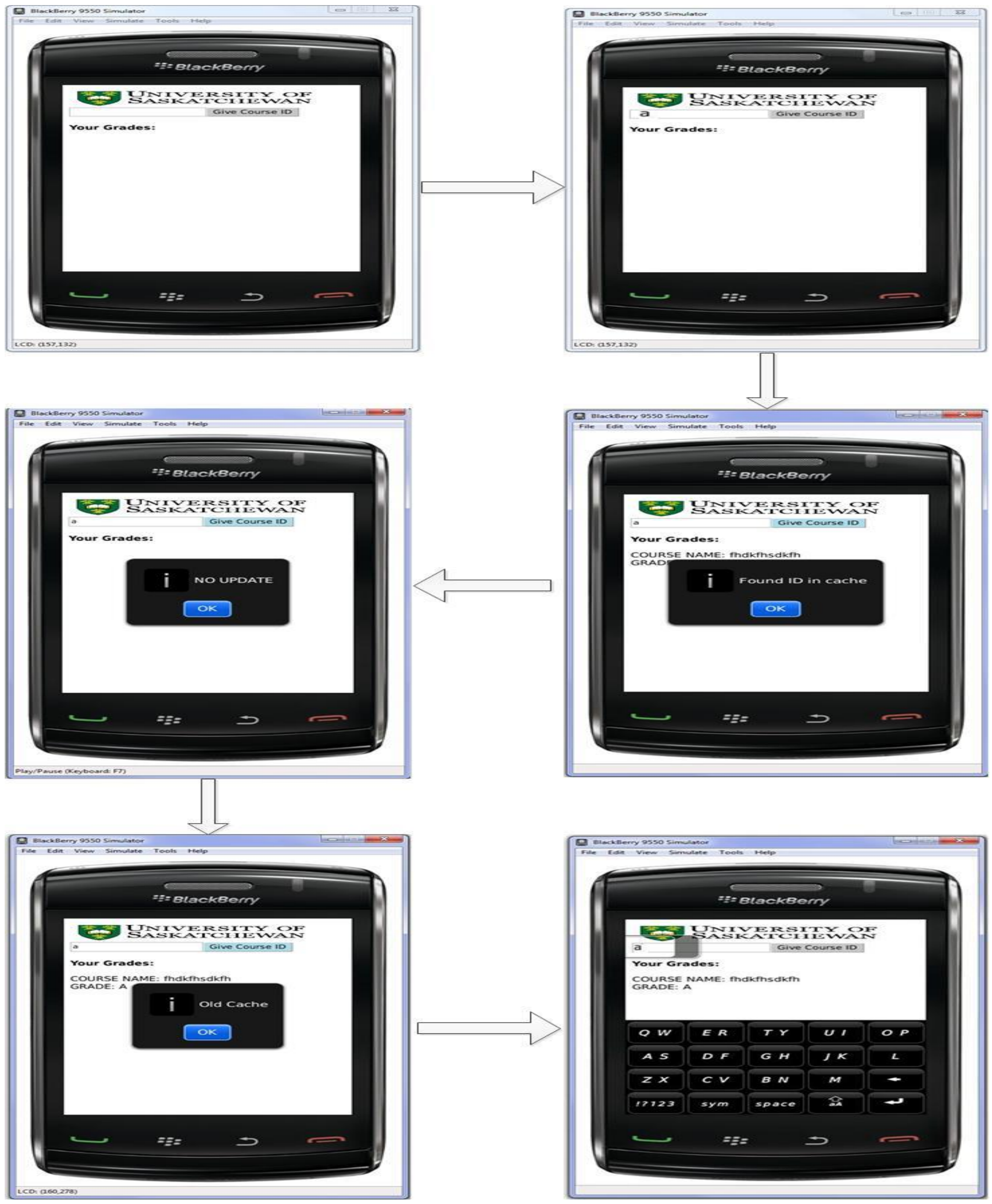

BlackBerry 9550 Simulator
File Edit View Simulate Tools Help
BlackBerry
UNIVERSITY OF SASKATCHEWAN
Give Course ID
Your Grades:
LCD: (157,132)
a
YES
OK
LCD: (170,93)

Your Grades:
COURSE NAME: social computing
GRADE: B

Your Grades:
COURSE NAME: fhdkfhsdkfh
GRADE: A

Figure 5-6(b). Condition 2

***Condition 3.* When the resource (requested grade) is found in the client cache and there is no update in any of the resources (grades):**


BlackBerry 9550 Simulator
File Edit View Simulate Tools Help
BlackBerry
UNIVERSITY OF SASKATCHEWAN
Give Course ID
Your Grades:
OK
LCD: (157,132)
Play/Pause (Keyboard: F7)
LCD: (160,278)

Found ID in cache

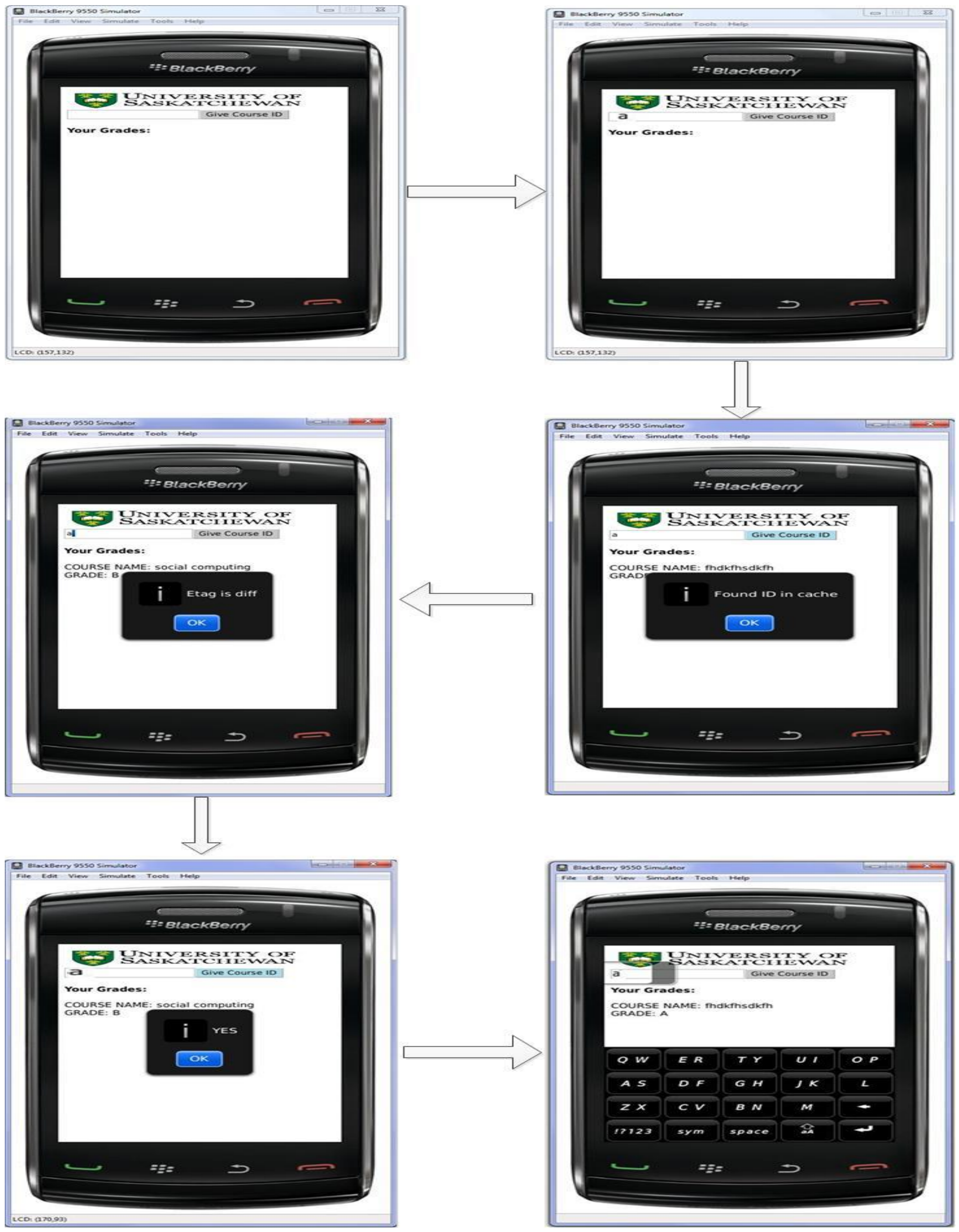
NO UPDATE

COURSE NAME: fhdkfhsdkfh
GRADE: A

Old Cache

Figure 5-6(c). Condition 3

Condition 4. **When the resource (requested grade) is found in the client cache and there is update in any of the resources (grades):**


BlackBerry 9550 Simulator
File Edit View Simulate Tools Help
BlackBerry
UNIVERSITY OF SASKATCHEWAN
Give Course ID
OK
LCD: (157,132)
LCD: (170,93)

Your Grades:

a

Your Grades:

a

Your Grades:

COURSE NAME: fhdkfhsdkfh

GRAD

Found ID in cache

a

Your Grades:

COURSE NAME: social computing

GRADE: B

Etag is diff

a

Your Grades:

COURSE NAME: social computing

GRADE: B

YES

a

Your Grades:

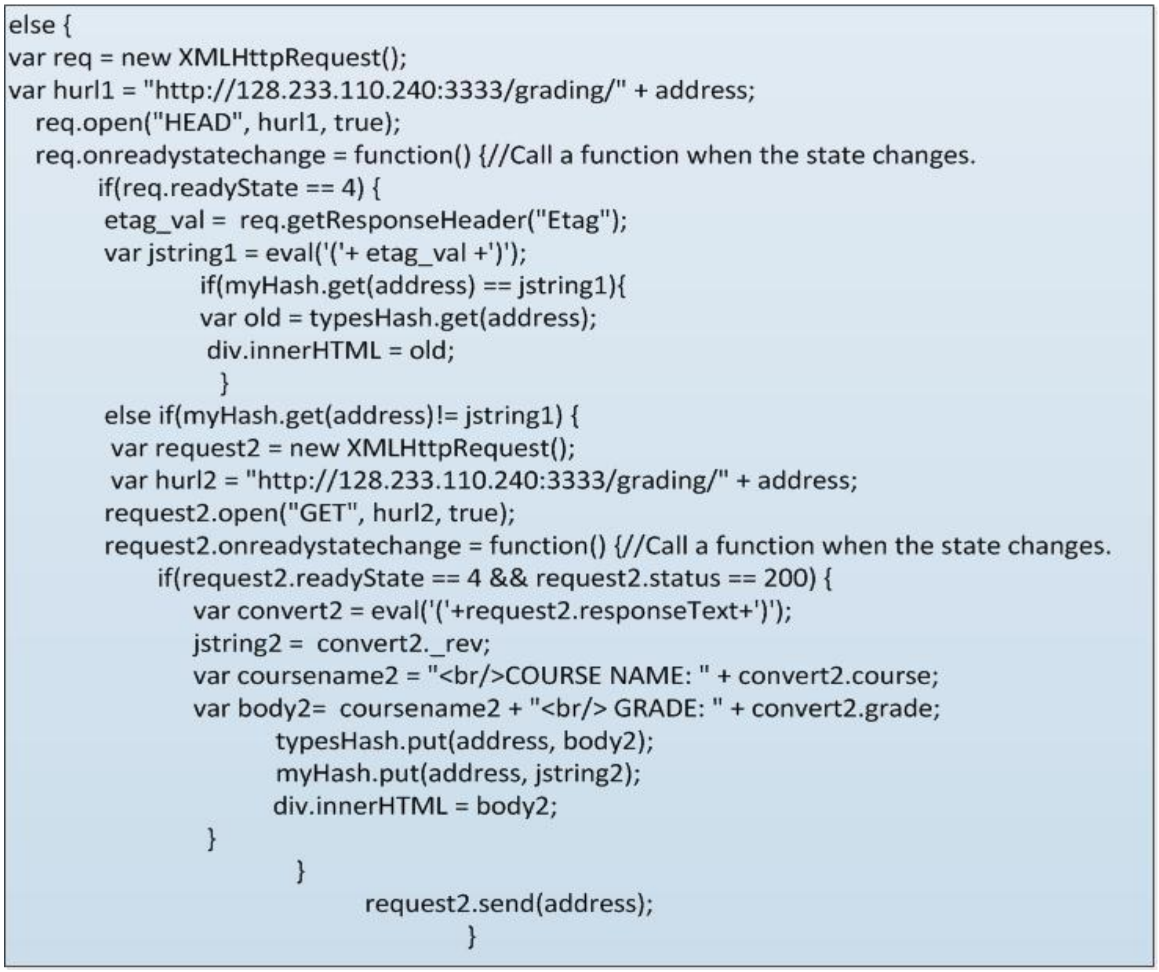
COURSE NAME: fhdkfhsdkfh

GRADE: A

Figure 5-6(d). Condition 4

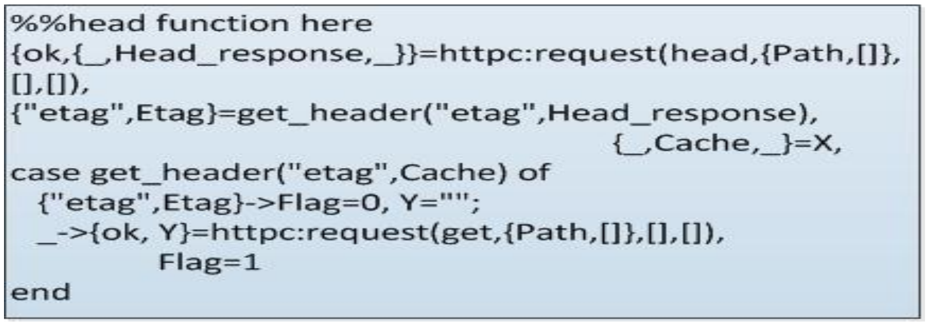
5.2 Caching Policy

For specifying the caching policy, using information in the meta tags [48] or in the headers is not considered in this research. However, everything which is cacheable is cached since HTTP GET request has been used as the request. Then, the HTTP HEAD request is used whenever there is a need for accessing the resource for cache validation. Figure 5-7 and figure 5-8 show the code snippets of the cache validation functions for the client using JavaScript and the proxy server using Erlang respectively. The Erlang proxy server is based on some previous research on REST and proxy server caching done by Lomotey [30].



```
else {
var req = new XMLHttpRequest();
var hurl1 = "http://128.233.110.240:3333/grading/" + address;
   req.open("HEAD", hurl1, true);
   req.onreadystatechange = function() {//Call a function when the state changes.
          if(req.readyState == 4) {
           etag_val =  req.getResponseHeader("Etag");
           var jstring1 = eval('('+ etag_val +')');
                      if(myHash.get(address) == jstring1){
                      var old = typesHash.get(address);
                       div.innerHTML = old;
                        }
           else if(myHash.get(address)!= jstring1) {
            var request2 = new XMLHttpRequest();
            var hurl2 = "http://128.233.110.240:3333/grading/" + address;
           request2.open("GET", hurl2, true);
           request2.onreadystatechange = function() {//Call a function when the state changes.
                 if(request2.readyState == 4 && request2.status == 200) {
                     var convert2 = eval('('+request2.responseText+')');
                     jstring2 =  convert2._rev;
                     var coursename2 = "<br/>COURSE NAME: " + convert2.course;
                     var body2=  coursename2 + "<br/> GRADE: " + convert2.grade;
                             typesHash.put(address, body2);
                             myHash.put(address, jstring2);
                             div.innerHTML = body2;
                      }
                               }
                                        request2.send(address);
                                                  }
```

Figure 5-7. Code snippet of the cache validation function for the client



```
%%head function here
{ok,{_,Head_response,_}}=httpc:request(head,{Path,[]},
[],[]),
{"etag",Etag}=get_header("etag",Head_response),
                                        {_,Cache,_}=X,
case get_header("etag",Cache) of
  {"etag",Etag}->Flag=0, Y="";
  _->{ok, Y}=httpc:request(get,{Path,[]},[],[]),
          Flag=1
end
```

Figure 5-8. Code snippet for the cache validation function for the proxy server [30]

5.3 Cache Data Structure

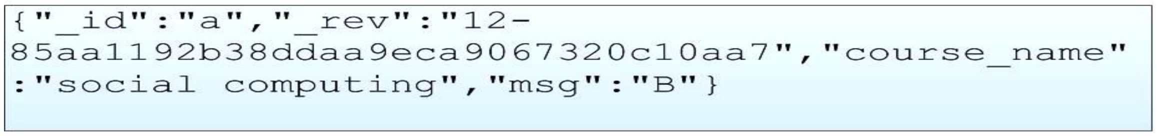
For the cached data which are JSON objects, the mobile client has its own cache which is a key/value pair storage built in JavaScript where the key of the storages are the resource URIs. In the proxy server, it has DETS [13] and ETS [13] tables built in Erlang which work as the database for the cached data where the key of the storages are the resource URIs as well.

5.4 State change messages using the Proxy server

To get the state changes, I implemented two different proxy servers using the Erlang programming language.

Proxy server A:

The mobile client gets the response back for a particular resource that has been updated from the proxy server through a JSON message. Figure 5-9 shows the response from the proxy server for the resource “b” whether there is any update or no update.

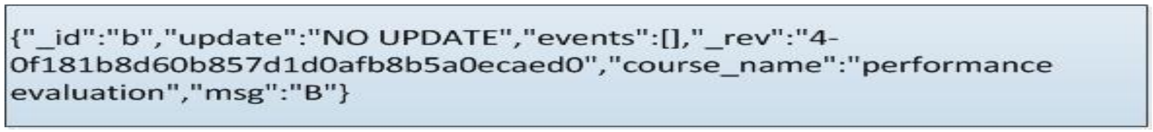


```
{"_id":"a","_rev":"12-
85aa1192b38ddaa9eca9067320c10aa7","course_name"
:"social computing","msg":"B"}
```

Figure 5-9. Response from the proxy server for the resource “b” whether there is any update or no update

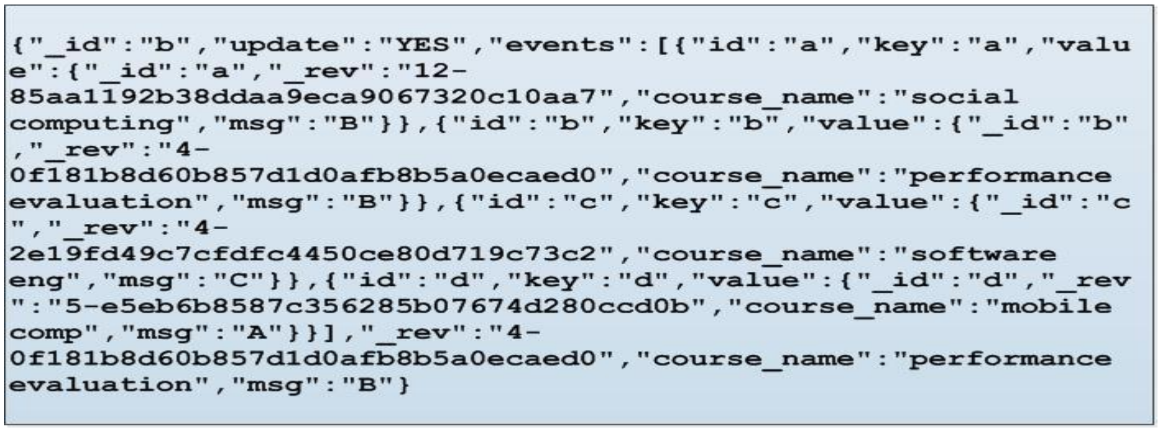
Proxy server B:

The mobile client gets the response back for a particular resource with all other resources that have been updated from the proxy server through a JSON message. Figure 5-10 shows the response from the proxy server for the resource “b” if there is no update and Figure 5-11 shows the response from the proxy server for the resource “b” if there is an update.



```
{"_id":"b","update":"NO UPDATE","events":[],"_rev":"4-
0f181b8d60b857d1d0afb8b5a0ecaed0","course_name":"performance
evaluation","msg":"B"}
```

Figure 5-10. Response from the proxy server for the resource “b” if there is no update



```
{"_id":"b","update":"YES","events":[{"id":"a","key":"a","valu
e":{"_id":"a","_rev":"12-
85aa1192b38ddaa9eca9067320c10aa7","course_name":"social
computing","msg":"B"}},{"id":"b","key":"b","value":{"_id":"b"
,"_rev":"4-
0f181b8d60b857d1d0afb8b5a0ecaed0","course_name":"performance
evaluation","msg":"B"}},{"id":"c","key":"c","value":{"_id":"c
","_rev":"4-
2e19fd49c7cfdfc4450ce80d719c73c2","course_name":"software
eng","msg":"C"}},{"id":"d","key":"d","value":{"_id":"d","_rev
":"5-e5eb6b8587c356285b07674d280ccd0b","course_name":"mobile
comp","msg":"A"}}],"_rev":"4-
0f181b8d60b857d1d0afb8b5a0ecaed0","course_name":"performance
evaluation","msg":"B"}
```

Figure 5-11. Response from the proxy server for the resource “b” if there is an update

The proxy server is responsible for consuming the WS, getting the state changes and delivering the service result to the mobile client.

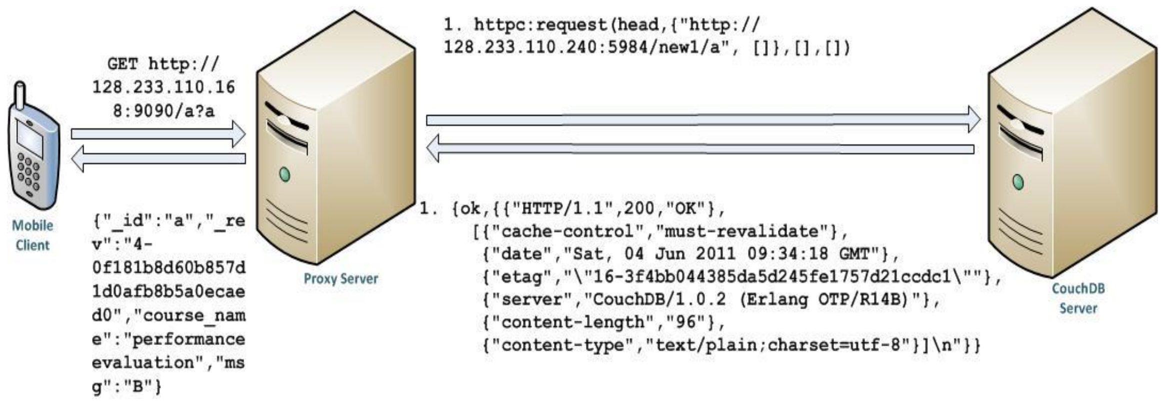
5.5 Workflow of the Proxy server

The workflows of the state propagations for the two different proxies are described below.

Proxy server A:

When there is no update in the resource: The GradeView application issues a GET request to the proxy server to know the grade of a course. The proxy server then sends ***Request 1*** which is a

HEAD request for the requested grade for a particular course that the client application has asked for to the REST Web Server and gets back ***Response 1*** which has the ETAG value for the grade. From the ETAG value, the proxy server gets to know if any state change happened to the requested grade or not by matching this value with its previous value which is kept in proxy cache. If not then the proxy server sends the response in JSON format for the grade that the client asked for from proxy cache. Figure 5-12(a) shows the workflow when there is no update in the grades.



```
GET http://
128.233.110.16
8:9090/a?a
```

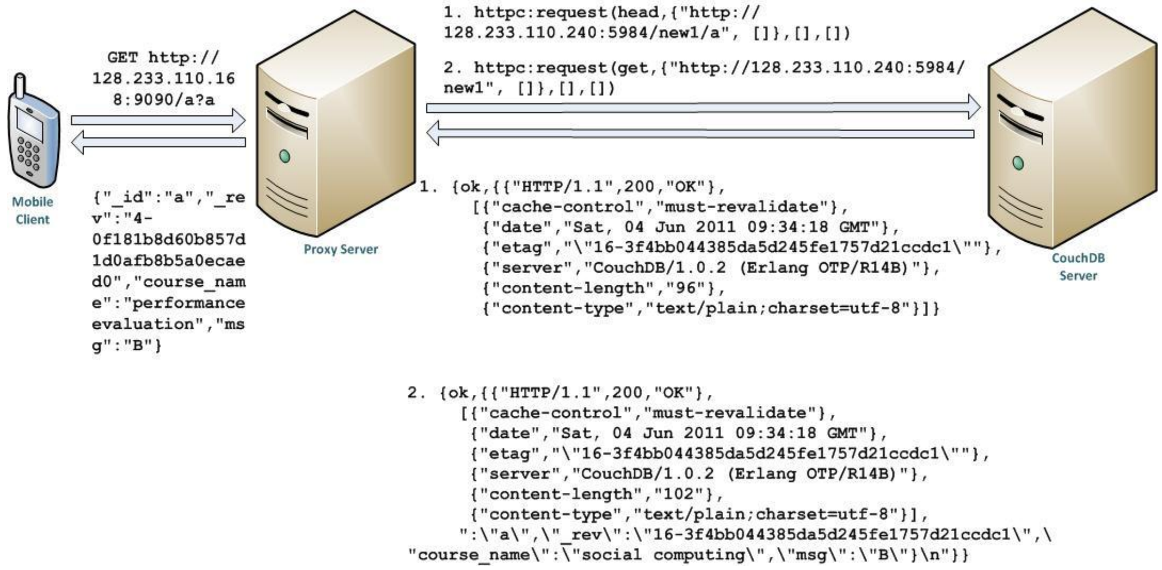
```
{"_id":"a","_re
v":"4-
0f181b8d60b857d
1d0afb8b5a0ecae
d0","course_nam
e":"performance
evaluation","ms
g":"B"}
```

```
1. httpc:request(head,{"http://
128.233.110.240:5984/new1/a", []},[],[])
```

```
1. {ok,{{"HTTP/1.1",200,"OK"},
    [{"cache-control","must-revalidate"},
     {"date","Sat, 04 Jun 2011 09:34:18 GMT"},
     {"etag","\"16-3f4bb044385da5d245fe1757d21ccdc1\""},
     {"server","CouchDB/1.0.2 (Erlang OTP/R14B)"},
     {"content-length","96"},
     {"content-type","text/plain;charset=utf-8"}]\n"}}
```

Figure 5-12(a). State propagation when there is no update

When there is an update in the resource: The GradeView application issues a GET request to the proxy server. The proxy server then sends ***Request 1*** to the REST Web server which is a HEAD request for the resource which is the grade of a course that the client asked for and gets back ***Response 1*** which has the ETAG value for the grade. From the ETAG value, the proxy server gets to know if any state change happened to the requested resource or not by matching this value with its previous value which is kept in proxy cache. If yes, then the proxy server sends ***Request 2*** which is a GET request to the REST Web server for the response of the grade that the client asked for and gets back ***Response 2.*** Then the proxy server sends the response in JSON format for the resource that the client asked for from the proxy cache. Figure 5-12(b) shows the workflow when there is an update in the grade.



```
GET http://
128.233.110.16
8:9090/a?a
```

```
{"_id":"a","_re
v":"4-
0f181b8d60b857d
1d0afb8b5a0ecae
d0","course_nam
e":"performance
evaluation","ms
g":"B"}
```

```
1. httpc:request(head,{"http://
128.233.110.240:5984/new1/a", []},[],[])

2. httpc:request(get,{"http://128.233.110.240:5984/
new1", []},[],[])
```

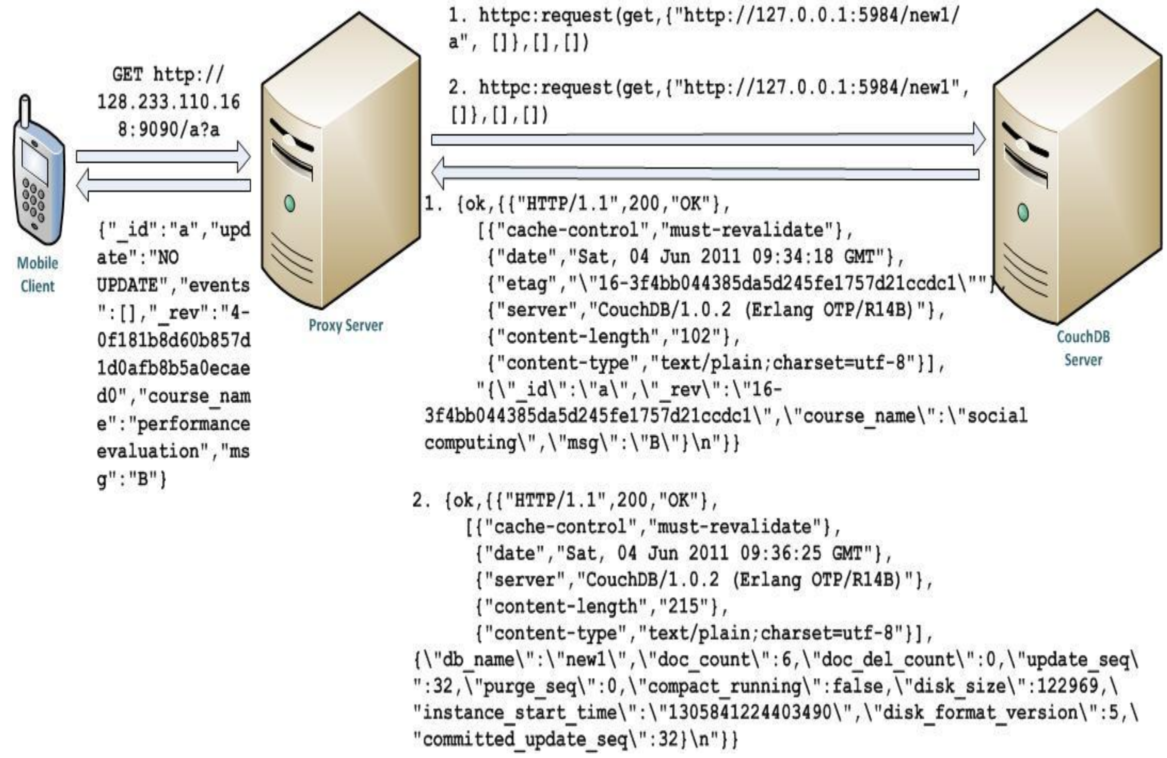
```
1. {ok,{{"HTTP/1.1",200,"OK"},
    [{"cache-control","must-revalidate"},
     {"date","Sat, 04 Jun 2011 09:34:18 GMT"},
     {"etag","\"16-3f4bb044385da5d245fe1757d21ccdc1\""},
     {"server","CouchDB/1.0.2 (Erlang OTP/R14B)"},
     {"content-length","96"},
     {"content-type","text/plain;charset=utf-8"}]}
```

```
2. {ok,{{"HTTP/1.1",200,"OK"},
    [{"cache-control","must-revalidate"},
     {"date","Sat, 04 Jun 2011 09:34:18 GMT"},
     {"etag","\"16-3f4bb044385da5d245fe1757d21ccdc1\""},
     {"server","CouchDB/1.0.2 (Erlang OTP/R14B)"},
     {"content-length","102"},
     {"content-type","text/plain;charset=utf-8"}],
    ":\"a\",\"_rev\":\"16-3f4bb044385da5d245fe1757d21ccdc1\",\
"course_name\":\"social computing\",\"msg\":\"B\"}\n"}}
```

Figure 5-12(b). State propagation when there is an update in the resource

Proxy server B:

When there is no update in the resources: The GradeView application issues a GET request to the proxy server. The proxy server then sends ***Request 1*** to the REST Web server which is for the response of the resource which is the grade of a course that the client asked for and gets back ***Response 1***. Then the proxy server sends ***Request 2*** to the server and gets ***Response 2***. This one contains the information about the status of the whole document. By analysing the last value of the response which says **"committed_update_seq",** proxy server gets to know if any state change happened to any of the grades or not by matching this value with its previous value which is kept in proxy's "record field". If not then the proxy server sends the response from the proxy cache in JSON format for the grade that the client asked for. Figure 5-13(a) shows the workflow when there is no update in the grades.



```
GET http://
128.233.110.16
8:9090/a?a
```

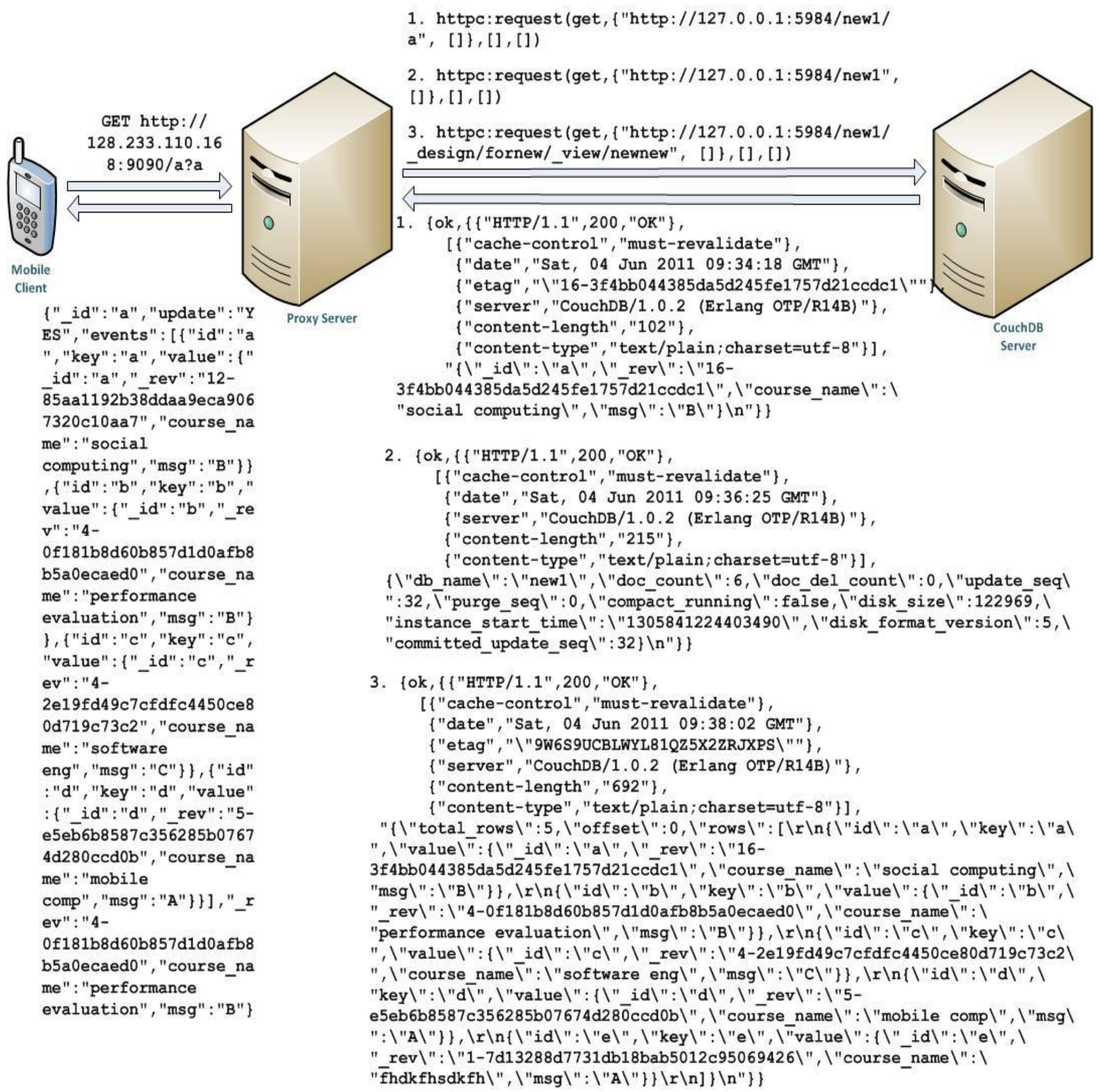
```
{"_id":"a","upd
ate":"NO
UPDATE","events
":[],"_rev":"4-
0f181b8d60b857d
1d0afb8b5a0ecae
d0","course_nam
e":"performance
evaluation","ms
g":"B"}
```

```
1. httpc:request(get,{"http://127.0.0.1:5984/new1/
a", []},[],[])

2. httpc:request(get,{"http://127.0.0.1:5984/new1",
[]},[],[])
```

```
1. {ok,{{"HTTP/1.1",200,"OK"},
     [{"cache-control","must-revalidate"},
      {"date","Sat, 04 Jun 2011 09:34:18 GMT"},
      {"etag","\"16-3f4bb044385da5d245fe1757d21ccdc1\""},
      {"server","CouchDB/1.0.2 (Erlang OTP/R14B)"},
      {"content-length","102"},
      {"content-type","text/plain;charset=utf-8"}],
     "{\"_id\":\"a\",\"_rev\":\"16-
3f4bb044385da5d245fe1757d21ccdc1\",\"course_name\":\"social
computing\",\"msg\":\"B\"}\n"}}

2. {ok,{{"HTTP/1.1",200,"OK"},
     [{"cache-control","must-revalidate"},
      {"date","Sat, 04 Jun 2011 09:36:25 GMT"},
      {"server","CouchDB/1.0.2 (Erlang OTP/R14B)"},
      {"content-length","215"},
      {"content-type","text/plain;charset=utf-8"}],
{\"db_name\":\"new1\",\"doc_count\":6,\"doc_del_count\":0,\"update_seq\
":32,\"purge_seq\":0,\"compact_running\":false,\"disk_size\":122969,\
"instance_start_time\":\"1305841224403490\",\"disk_format_version\":5,\
"committed_update_seq\":32}\n"}}
```

Figure 5-13(a). State propagation when there is no update

When there are updates in the resources: The GradeView application issues a GET request to the proxy server. The proxy server then sends ***Request 1*** to the REST Web server which is for the response of the grade that the client asked for and gets back ***Response 1***. Then the proxy server sends ***Request 2*** to the REST Web server and gets ***Response 2***. This one contains the information about the status of the whole document. By analysing the last value of the response which says **"committed_update_seq",** proxy server gets to know if any state change happened to any of the grades or not by matching this value with its previous value which is kept in proxy's "record field". If yes then the proxy server updates its status with the recent value of **"committed_update_seq" and** sends ***Request 3*** to the REST Web server and gets back ***Response 3*** which contains all the grades where any update happened. Then the proxy server sends the response in JSON format which includes the response for the grade that the client asked for and all other updated grades. Figure 5-13(b) shows the workflow when there are updates in resources.



```
GET http://
128.233.110.16
8:9090/a?a
```

```
1. httpc:request(get,{"http://127.0.0.1:5984/new1/
a", []},[],[])

2. httpc:request(get,{"http://127.0.0.1:5984/new1",
[]},[],[])

3. httpc:request(get,{"http://127.0.0.1:5984/new1/
_design/fornew/_view/newnew", []},[],[])
```

```
1. {ok,{{"HTTP/1.1",200,"OK"},
     [{"cache-control","must-revalidate"},
      {"date","Sat, 04 Jun 2011 09:34:18 GMT"},
      {"etag","\"16-3f4bb044385da5d245fe1757d21ccdc1\""},
      {"server","CouchDB/1.0.2 (Erlang OTP/R14B)"},
      {"content-length","102"},
      {"content-type","text/plain;charset=utf-8"}],
     "{\"_id\":\"a\",\"_rev\":\"16-
3f4bb044385da5d245fe1757d21ccdc1\",\"course_name\":\
"social computing\",\"msg\":\"B\"}\n"}}

2. {ok,{{"HTTP/1.1",200,"OK"},
     [{"cache-control","must-revalidate"},
      {"date","Sat, 04 Jun 2011 09:36:25 GMT"},
      {"server","CouchDB/1.0.2 (Erlang OTP/R14B)"},
      {"content-length","215"},
      {"content-type","text/plain;charset=utf-8"}],
{\"db_name\":\"new1\",\"doc_count\":6,\"doc_del_count\":0,\"update_seq\
":32,\"purge_seq\":0,\"compact_running\":false,\"disk_size\":122969,\
"instance_start_time\":\"1305841224403490\",\"disk_format_version\":5,\
"committed_update_seq\":32}\n"}}

3. {ok,{{"HTTP/1.1",200,"OK"},
     [{"cache-control","must-revalidate"},
      {"date","Sat, 04 Jun 2011 09:38:02 GMT"},
      {"etag","\"9W6S9UCBLWYL81QZ5X2ZRJXPS\""},
      {"server","CouchDB/1.0.2 (Erlang OTP/R14B)"},
      {"content-length","692"},
      {"content-type","text/plain;charset=utf-8"}],
 "{\"total_rows\":5,\"offset\":0,\"rows\":[\r\n{\"id\":\"a\",\"key\":\"a\
",\"value\":{\"_id\":\"a\",\"_rev\":\"16-
3f4bb044385da5d245fe1757d21ccdc1\",\"course_name\":\"social computing\",\
"msg\":\"B\"}},\r\n{\"id\":\"b\",\"key\":\"b\",\"value\":{\"_id\":\"b\",\
"_rev\":\"4-0f181b8d60b857d1d0afb8b5a0ecaed0\",\"course_name\":\
"performance evaluation\",\"msg\":\"B\"}},\r\n{\"id\":\"c\",\"key\":\"c\
",\"value\":{\"_id\":\"c\",\"_rev\":\"4-2e19fd49c7cfdfc4450ce80d719c73c2\
",\"course_name\":\"software eng\",\"msg\":\"C\"}},\r\n{\"id\":\"d\",\
"key\":\"d\",\"value\":{\"_id\":\"d\",\"_rev\":\"5-
e5eb6b8587c356285b07674d280ccd0b\",\"course_name\":\"mobile comp\",\"msg\
":\"A\"}},\r\n{\"id\":\"e\",\"key\":\"e\",\"value\":{\"_id\":\"e\",\
"_rev\":\"1-7d13288d7731db18bab5012c95069426\",\"course_name\":\
"fhdkfhsdkfh\",\"msg\":\"A\"}}\r\n]}\n"}}
```

```
{"_id":"a","update":"Y
ES","events":[{"id":"a
","key":"a","value":{"
_id":"a","_rev":"12-
85aa1192b38ddaa9eca906
7320c10aa7","course_na
me":"social
computing","msg":"B"}}
,{"id":"b","key":"b","
value":{"_id":"b","_re
v":"4-
0f181b8d60b857d1d0afb8
b5a0ecaed0","course_na
me":"performance
evaluation","msg":"B"}
},{"id":"c","key":"c",
"value":{"_id":"c","_r
ev":"4-
2e19fd49c7cfdfc4450ce8
0d719c73c2","course_na
me":"software
eng","msg":"C"}},{"id"
:"d","key":"d","value"
:{"_id":"d","_rev":"5-
e5eb6b8587c356285b0767
4d280ccd0b","course_na
me":"mobile
comp","msg":"A"}}],"_r
ev":"4-
0f181b8d60b857d1d0afb8
b5a0ecaed0","course_na
me":"performance
evaluation","msg":"B"}
```

Figure 5-13(b). State propagation when there are updates in the resources

CHAPTER 6
EXPERIMENTS

For the experimental setup, a BlackBerry 9550 Simulator, a BlackBerry Playbook tablet device and an Android Honeycomb tablet Emulator (Android 3:2 Emulator) are used as the mobile clients to simplify the evaluation. The client-side of the GradeView app is an embedded browser framework which is implemented using WebWorks for the BlackBerry platform and WebView for the Android platform. The client application is implemented using JavaScript. A proxy server is used as the intermediary between the mobile client and the RESTful Web server to handle the numerous concurrent requests. The Erlang-based proxy server and the CouchDB RESTful Web Server (which is a document oriented Non-Relational Database Management Server) are hosted on servers that are identical to the Amazon EC2 cloud provider. Figure 6-1 shows the experimental setup.

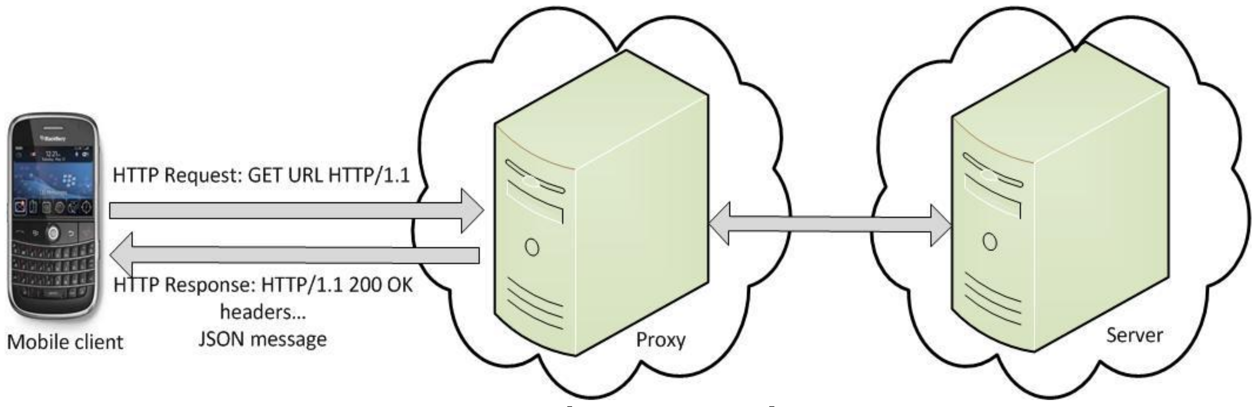

HTTP Request: GET URL HTTP/1.1
HTTP Response: HTTP/1.1 200 OK
headers...
JSON message
Mobile client
Proxy
Server

Figure 6-1. The Experimental Setup

For all the experiments, the BlackBerry 9550 Simulator and Android Honeycomb tablet Emulator are hosted on a 4GB with an Intel®Core™ i5 CPU 650 that ran at 3.20GHz. A gigabit Ethernet connection is used to link the RESTful services and the proxy server. The REST Web server, which is hosting the services and the proxy server ran on two different servers that use Intel®Xeon® CPU E5506 at 2.13GHz and accessed 16 GB of RAM.

6.1 Experiment Goals

Goal 1. What is the maximum request/ response rate when the proxy server and the mobile client communicate? This is to determine how fast the mobile client and proxy server can efficiently communicate.

Goal 2. How the state changes can be detected and propagated to the proxy server and the mobile client?

Goal 3. How multiple devices of a single user can be supported by the cache efficiently?

The experiment goals and the corresponding experiments are shown in Table 6-1. Figure 6-2 shows the experiment goals in the proposed system.

Table 6-1. Experiments and Goals

Goal	**Experiment**
What is the maximum request/ response rate when the proxy server and the mobile client communicate? This is to determine how fast the mobile client and proxy server can efficiently communicate.	Overhead Calculation
How the state changes can be detected and propagated to the proxy server and the mobile client?	State propagation test
How multiple devices of a single user can be supported by the cache efficiently?	Scalability Test

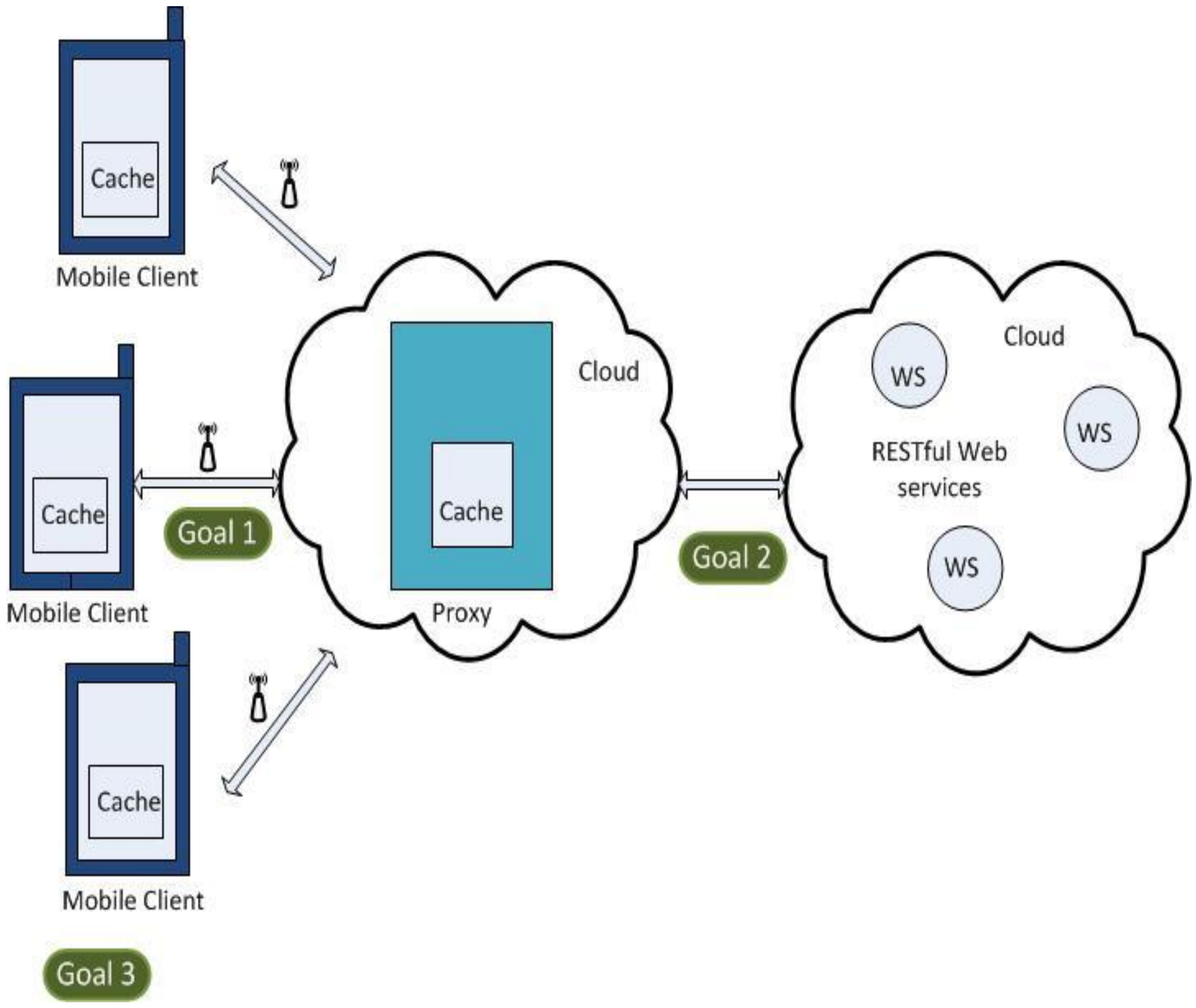


Figure 6-2. Experiment goals

6.2 Experiments

The following is a list of experiments that is conducted to evaluate the design of the proxy server and the mobile client according to the architecture.

Experiment 1: Overhead Calculation

Experiment goal: experiment 1 is to measure the general overhead of the system to test Goal 1 which is **to determine what is the maximum request/ response rate when the proxy server and the client communicate.**

Experiment 2: State propagation test

Experiment goal: experiment 2 is to test Goal 2 which is **to determine the delay between the state change occurring and the cache being notified and propagated to the proxy server and the mobile client.**

Experiment 3: Scalability Test

Experiment goal: experiment 3 is to test the scalability and reliability of the system to test Goal 3 which is **to show how multiple devices of a single user can be supported by the cache efficiently.**

6.3 Experiment Setup

For conducting experiment 1 and experiment 2, three different setups are deployed.

Setup A. The first experiment is conducted without cache in the BlackBerry WebWorks application that consumes the REST WS directly (no proxy server) using a stable high speed connection.

Setup B. For the second experiment (with cache) the BlackBerry WebWorks application was extended with a cache component.

Setup C. In the third experiment (with proxy server) we used the setup of experiment two but included a proxy server (with cache).

Experiment 1: Overhead Calculation

a. For the first experiment I ran the system with a BlackBerry WebWorks application that consumes directly (no proxy server) using a stable high speed connection and no cache and calculated the overhead. To measure the overhead, request/ response time is calculated for the services as the number of request is increased from 1 to 30 and 1 to 150 when the BlackBerry 9550 Simulator and BlackBerry Playbook tablet device are used as the mobile clients respectively. Figure 6-3 describes the first setup for the overhead calculation where the mobile client is consuming the RESTful WS from the REST Web server.

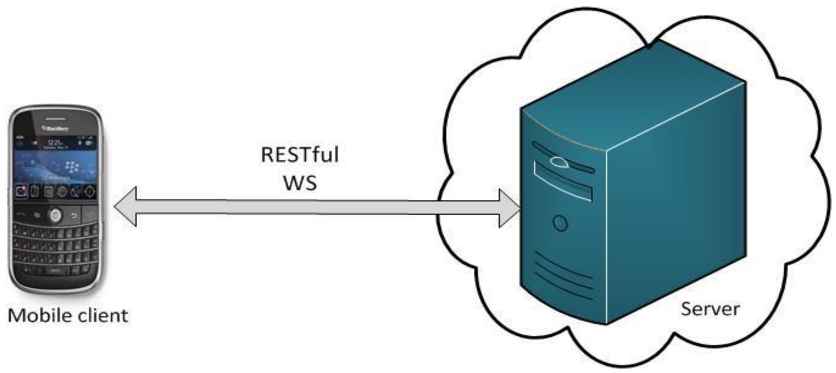


Figure 6-3. Consume RESTful WS from the REST Web server

b. For the second experiment I ran the system with a BlackBerry WebWorks application that consumes directly (no proxy server) using a stable high speed connection and calculated the overhead where the BlackBerry WebWorks application was extended with a cache component. To measure the overhead, request/ response time is calculated for the services as the number of request is increased from 1 to 30 and 1 to 150 when the BlackBerry 9550 Simulator and BlackBerry Playbook tablet device are used as the mobile clients respectively. Figure 6-4 describes the second setup for the overhead calculation where the mobile client is consuming the RESTful WS from the REST Web server.

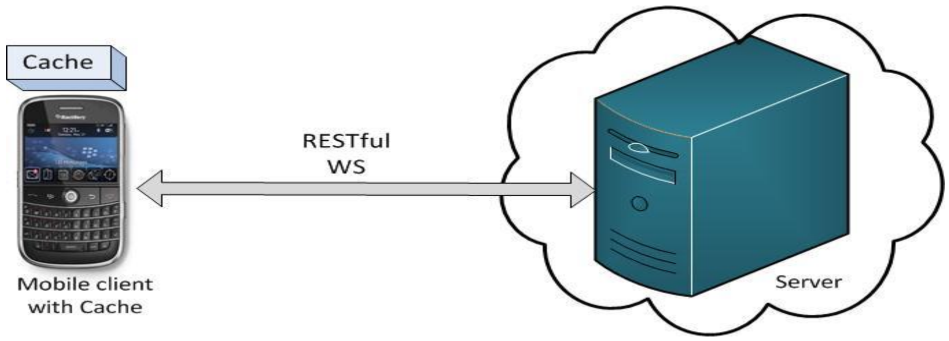


Figure 6-4. Consume RESTful WS from the REST Web server (mobile client with cache)

c. For the second experiment I ran the system with a BlackBerry WebWorks application that consumes through proxy server using a stable high speed connection and calculated the overhead where the BlackBerry WebWorks application was extended with a cache component. The proxy server had its own caching component in it as well. To measure the overhead, request/ response time is calculated for the services as the number of request is increased from 1 to 30 and 1 to 150 when the BlackBerry 9550 Simulator and BlackBerry Playbook tablet device are used as the mobile clients respectively. Figure 6-5 describes the second setup for the overhead calculation where the mobile client is consuming the RESTful WS through a cloud hosted proxy server.

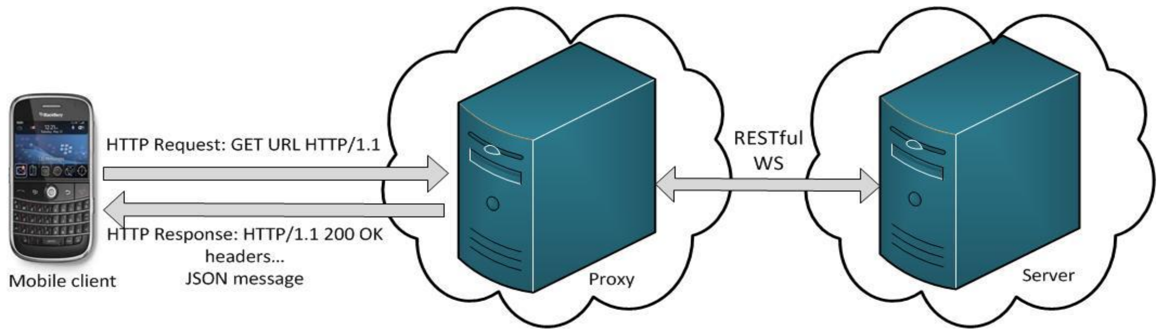

HTTP Request: GET URL HTTP/1.1
HTTP Response: HTTP/1.1 200 OK
headers...
JSON message
Mobile client
Proxy
RESTful
WS
Server

Figure 6-5. Consume RESTful WS through cloud hosted proxy server

Experiment 2: State Propagation test

a. For the first experiment I ran the system with a BlackBerry WebWorks application that consumes directly (no proxy server) using a stable high speed connection and no cache. To determine the delay between the state change occurring and the client being notified, maximum, minimum and average required time is calculated for the services as the number of request is be increased from 1 to 30 and 1 to 150 when the BlackBerry 9550 Simulator and BlackBerry Playbook tablet device are used as the mobile clients respectively. Figure 6-6 describes the first setup for the overhead calculation where the mobile client is consuming the RESTful WS from the REST Web server.

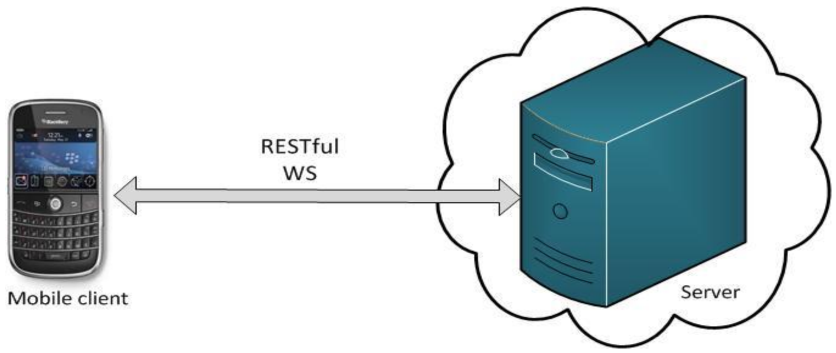


Figure 6-6. Consume RESTful WS from the REST Web server

b. For the second experiment I ran the system with a BlackBerry WebWorks application that consumes directly (no proxy server) using a stable high speed connection where the BlackBerry WebWorks application was extended with a cache component. To determine the delay between the state change occurring and the client being notified, maximum, minimum and average required time is calculated for the services as the number of request is be increased from 1 to 30 and 1 to 150 when the BlackBerry 9550 Simulator and BlackBerry Playbook tablet device are used as the mobile clients respectively.. Figure 6-7 describes the second setup for the overhead calculation where the mobile client is consuming the RESTful WS from the REST Web server.

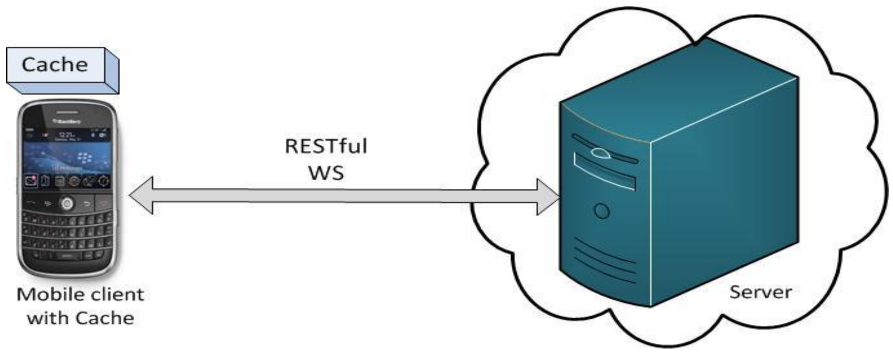


Figure 6-7. Consume RESTful WS from the REST Web server (mobile client with cache)

c. For the third experiment I ran the system with a BlackBerry WebWorks application that communicates through the proxy server using a stable high speed connection where the BlackBerry WebWorks application was extended with a cache component. The proxy server had its own cache component in it as well. For this experiment I used two different proxies; proxy server A and proxy server B with different state propagation mechanism as discussed in earlier chapters. To determine the delay between the state change occurring and the client being notified, maximum, minimum and average required time is calculated for the services as the number of request is be increased from 1 to 30 and 1 to 150 when the BlackBerry 9550 Simulator and BlackBerry Playbook tablet device are used as the mobile clients respectively. Figure 6-8 describes the second setup for the overhead calculation where the mobile client is consuming the RESTful WS through cloud hosted proxy server.

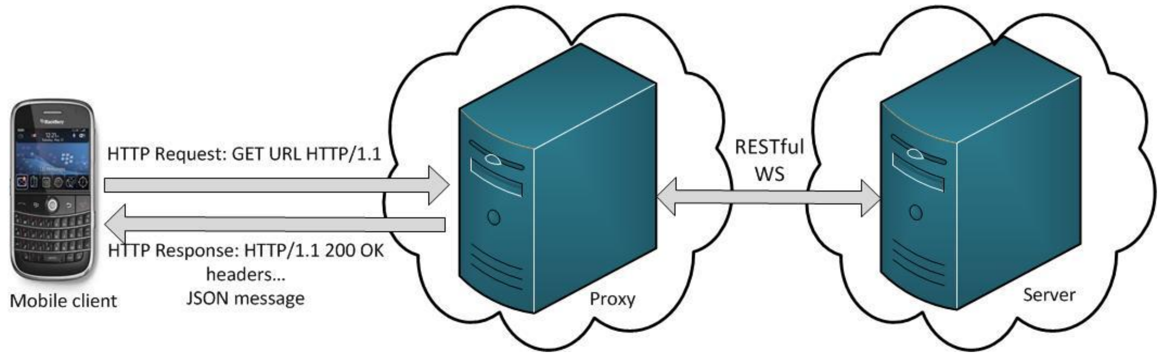

HTTP Request: GET URL HTTP/1.1
HTTP Response: HTTP/1.1 200 OK
headers...
JSON message
Mobile client
RESTful
WS
Proxy
Server

Figure 6-8. Consume RESTful WS through cloud hosted proxy server

Experiment 3: Scalability Test

a. To test the scalability of the entire system, in the first experiment, apache bench [5] load generator is configured to issue 10000 workloads as the number of concurrent request is changed from 10 to 550 and sent it to the REST Web server. The performance is determined based on how the concurrent requests the system is handling by calculating the throughput and time per request across all concurrent requests. And Figure 6-9 describes the first setup for the scalability test where Load Generator is sending requests to the REST Web server.

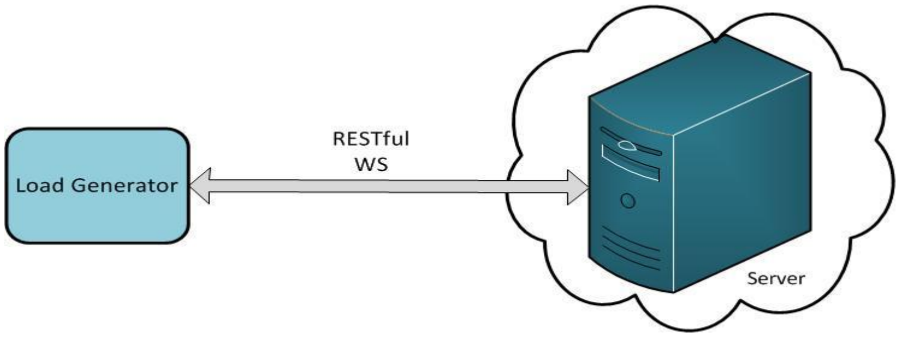


Figure 6-9. Load Generator sends requests to the REST Web server

b. In the second experiment, apache bench load generator generated 10000 workloads as the number of concurrent request is changed from 10 to 550 and sent it to the proxy server. For this experiment I used two different proxies; proxy server A and proxy server B with different state propagation mechanism as discussed in earlier chapters. The performance is determined based on how the concurrent requests the system is handling by calculating the throughput and time per request across all concurrent requests. Figure 6-10 describes the second setup for the scalability test where Load Generator is sending requests to the proxy server.

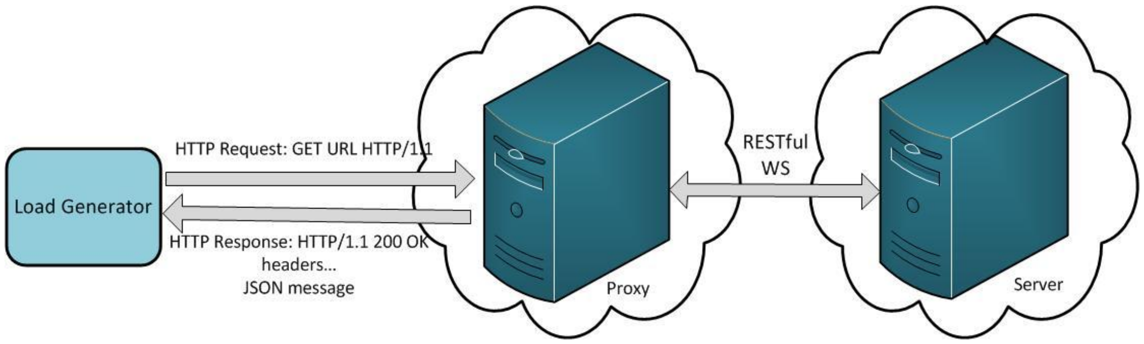

Load Generator
HTTP Request: GET URL HTTP/1.1
HTTP Response: HTTP/1.1 200 OK
headers...
JSON message
Proxy
RESTful
WS
Server

Figure 6-10. Load Generator sends requests to the proxy Server

6.4 Experiment Results and Discussion

Experiment 1: Overhead Calculation

For experiment 1, to calculate the overhead, the systems were evaluated based on different numbers of concurrent clients that each executed 30 read requests (for the BlackBerry 9550 Simulator) and 150 read requests (for the BlackBerry Playbook device) to a RESTful web service respectively. The tests are repeated five (5) times on each round starting from 1 request to 30 requests (for the BlackBerry 9550 Simulator) and 1 request to 150 requests (for the BlackBerry Playbook device). Table 6-2 shows a comparative analysis of the three setups based on the throughput in seconds for the services as the number of requests is changed from 1 to 30 for the BlackBerry 9550 Simulator. Table 6-3 shows a comparative analysis of the three setups based on the throughput in seconds for the services as the number of requests is changed from 1 to 150 for the BlackBerry Playbook device.

Table 6-2. Results comparing the three setups for the BlackBerry 9550 Simulator

Experiment Setup	Setup A (no cache and no proxy server)	Setup B (cache and no proxy server)	Setup C (proxy server and cache)
Average throughput for sending 30 requests (s)	107.6722341	117.1247076	237.2614898

Table 6-3. Results comparing the three setups for the BlackBerry Playbook device

Experiment Setup	Setup A (no cache and no proxy server)	Setup B (cache and no proxy server)	Setup C (proxy server and cache)
Average throughput for sending 150 requests (s)	398.93301	592.0737346	2652.800438

The results from table 6-2 show that Setup C improves the throughput for 120.3553142% than the Setup A and 102.5716816% than the Setup B under the same conditions when using the BlackBerry 9550 Simulator as the mobile device. The results from table 6-3 show that Setup C improves the throughput for 564.9739103% than the Setup A and 348.0523764% than the Setup B under the same conditions when using the BlackBerry Playbook device as the mobile device. Figure 6-11(a) shows the throughput in seconds for the services as the number of requests is changed from 1 to 30 for the 3 different models using the BlackBerry 9550 Simulator as the mobile client. Figure 6-11(b) shows the throughput in seconds for the services as the number of requests is changed from 1 to 150 for the 3 different models using the BlackBerry Playbook tablet device as the mobile client. Setup A (no cache and no proxy server), Setup B (cache in mobile client) and Setup C (proxy server and cache) for experiment 1 are represented in the graphs by the blue, red and green line respectively.

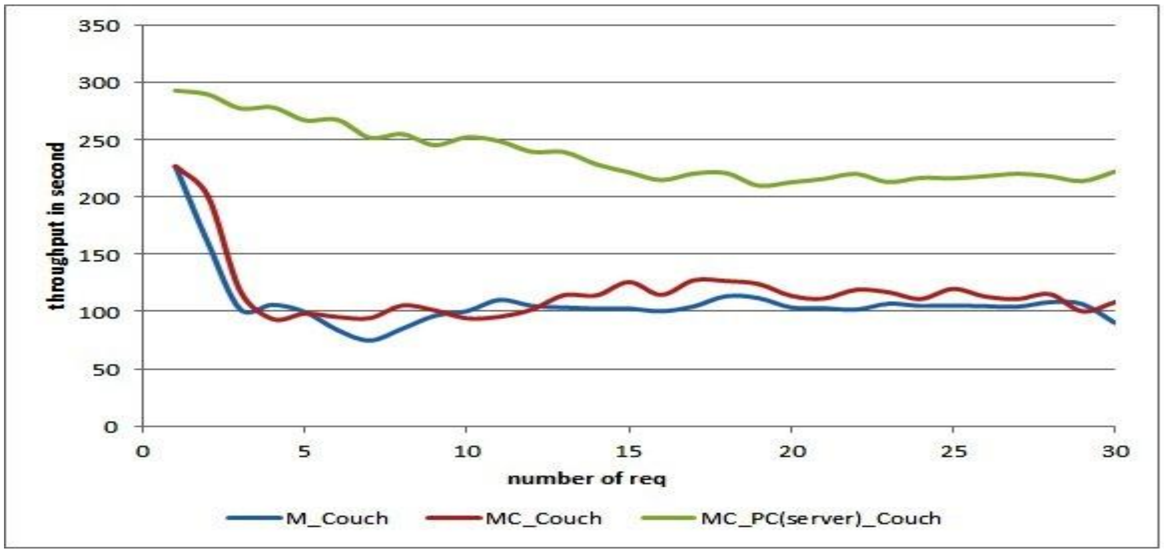


Figure 6-11(a). Throughput in second for the BlackBerry 9550 Simulator

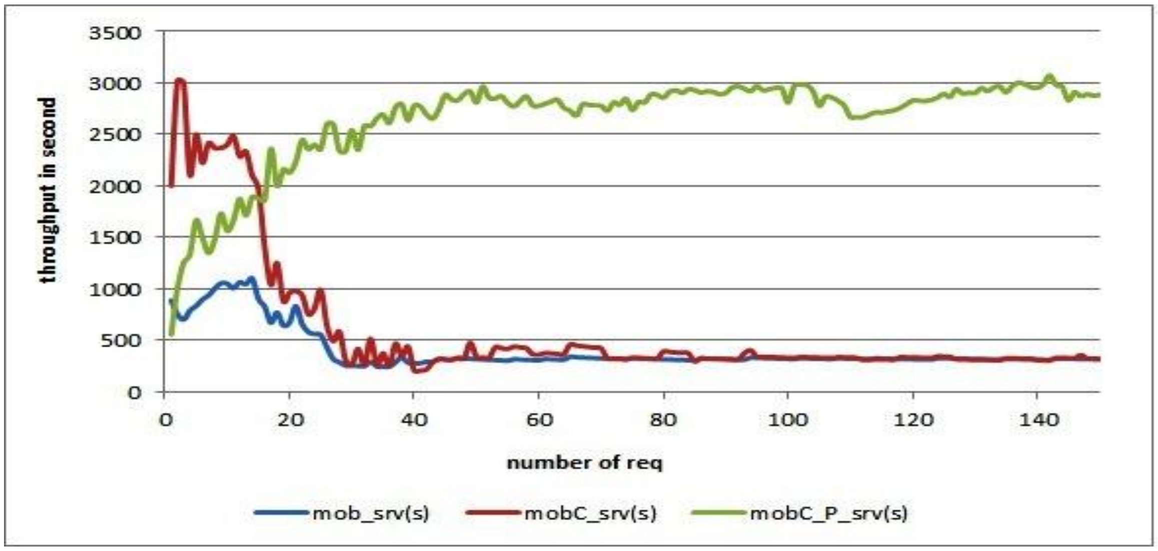


Figure 6-11(b). Throughput in second for the BlackBerry Playbook device

The graph shows decrement of throughput for model A and B after a while as number of requests increases from 1 to 30 and linear line in throughput for model C in Fig 6-11a. In Fig 6-11b, the graph shows an increment of throughput for model C (Fig 6-11a) and decrement of throughput after a while for model A and B in Fig 6-11b as number of requests increases from 1 to 150. The graphs, Fig 6-11a shows a drop at the beginning for model A and Fig 6-11b shows a spike at the beginning for model B respectively which is maybe while the CouchDB cache was being filled. For both Fig 6-11a and Fig 6-11b, model C performs better than model A and model B. Also high throughput is achieved using model C over model A and B. The result is best when the tablet device is used over the simulator.

Experiment 2: State Propagation test

For experiment 2, to determine the delay between the state change occurring and the client being notified, maximum, minimum and average required time is calculated based on different numbers of concurrent clients that each executed 30 read requests (for the BlackBerry 9550 Simulator) and 150 read requests (for the BlackBerry Playbook device) to a RESTful Web server respectively.

The tests are repeated five (5) times on each round starting from 1 request to 30 requests (for the BlackBerry 9550 Simulator) and 1 request to 150 requests (for the BlackBerry Playbook device). Table 6-4 and table 6-5 show the result of the three setups based on the minimum and maximum required response time in seconds for the services as the number of requests is changed from 1 to 30 using the proxy server A and proxy server B respectively.

Table 6-4. Request-response duration through proxy server A

Experiment Setup	Setup A (no cache and no proxy server)	Setup B (cache and no proxy server)	Setup C (proxy server A and cache)
Min request-response time for 30 requests (s)	0.110867	0.097067	0.0651
Max request-response time for 30 requests (s)	0.125633	0.139	0.076233

Table 6-5. Request-response duration through proxy server B

Experiment Setup	Setup A (no cache and no proxy server)	Setup B (cache and no proxy server)	Setup C (proxy server B and cache)
Min request-response time for 30 requests (s)	0.110867	0.097067	0.1144
Max request-response time for 30 requests (s)	0.125633	0.139	0.132633

Figure 6-12 and Figure 6-13 shows the minimum and maximum required time in seconds for the services as the number of requests is changed from 1 to 30 for the 3 different models using the BlackBerry 9550 Simulator as the mobile client respectively. Setup A (no cache and no proxy server), Setup B (cache in mobile client) and Setup C (proxy server and cache) for experiment 1 are represented in the graphs by the blue, red and green line respectively. Here I used proxy server A for the first part of the graph and proxy server B for the second part of the graph.

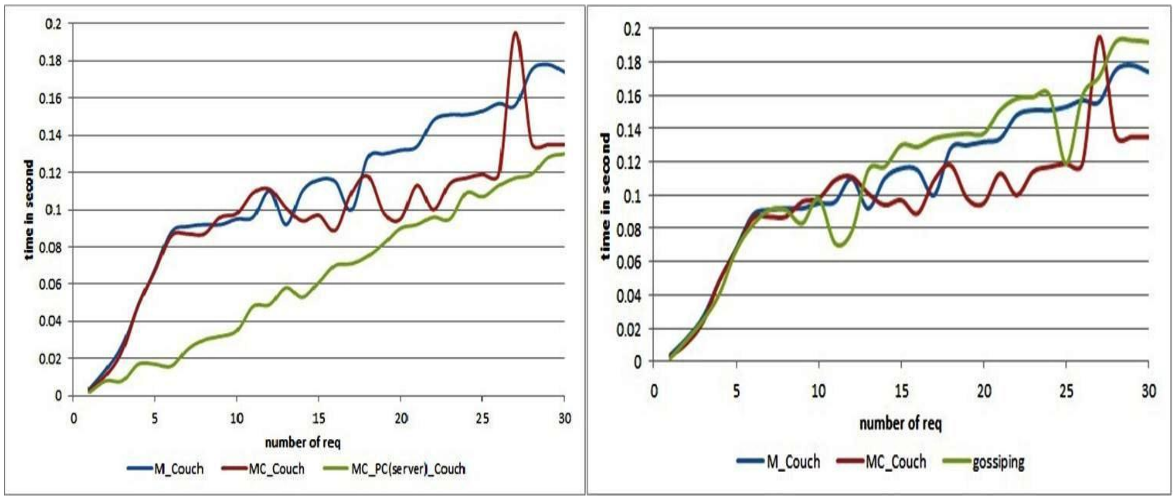


Figure 6-12. Minimum response times

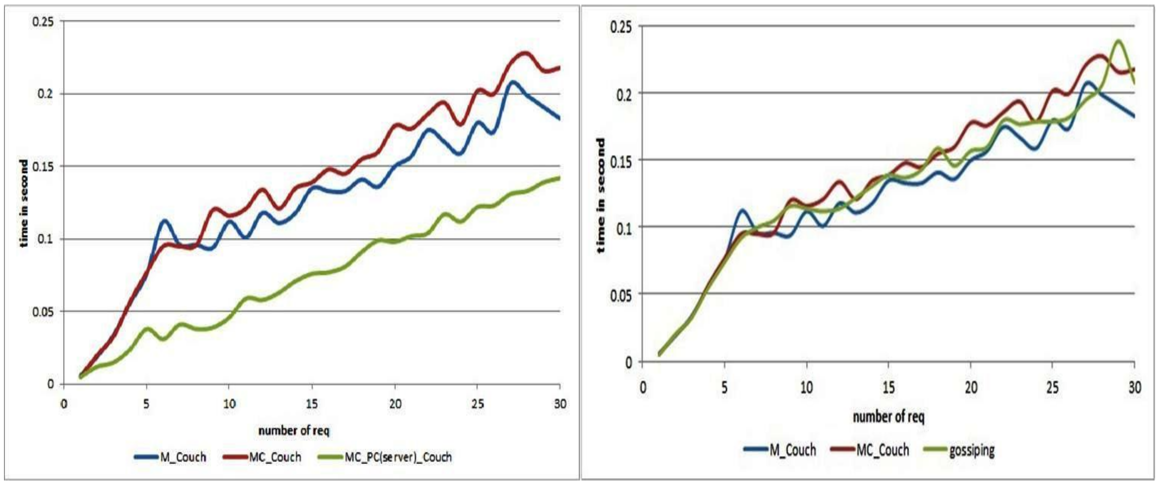


Figure 6-13. Maximum response times

Figure 6-12 and Figure 6-13 show an increment of required time as a function of the number of request for the model A, B and model C (as we used closed loop for multiple GET requests) when Proxy server A is used (the first part of the graphs). Here, the results show (from table 6-4, figure 6-12 and 6-13) the minimum and maximum response times are minimal as the number of requests is changed from 1 to 30 when model C (using proxy server A) is used which is represented by the green line. Though there is no notable performance increase when Proxy server B is used (the second part of the graphs) for model C in Figure 6-12, Figure 6-13 and table 6-5.

Figure 6-14 and figure 6-15 show the comparison of minimum and maximum response times for two different proxies. Proxy server A and Proxy server B (gossiping) are represented in the graphs by the blue and red line respectively.

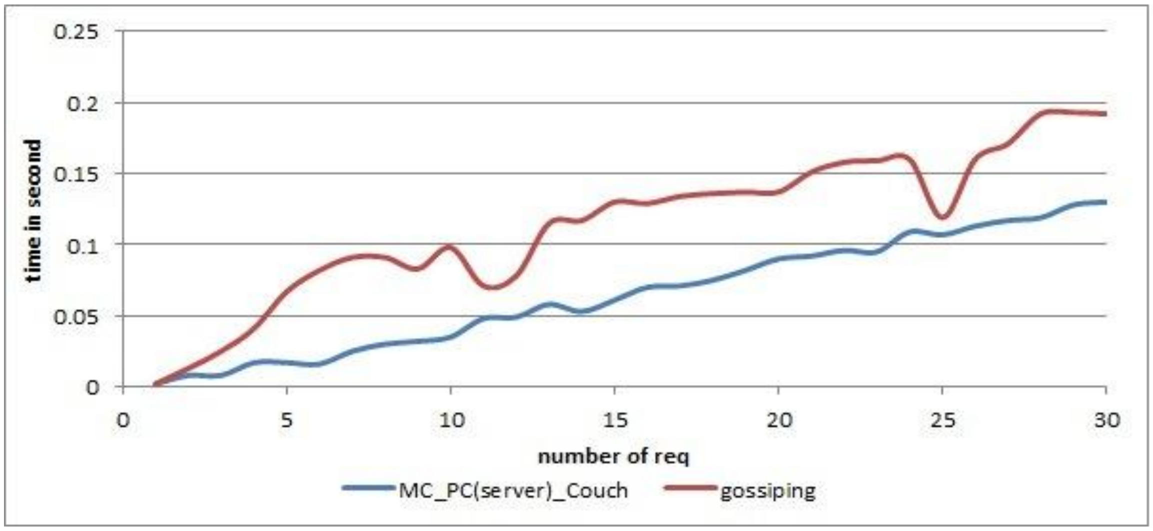


Figure 6-14. Comparison of minimum response times for two different proxies

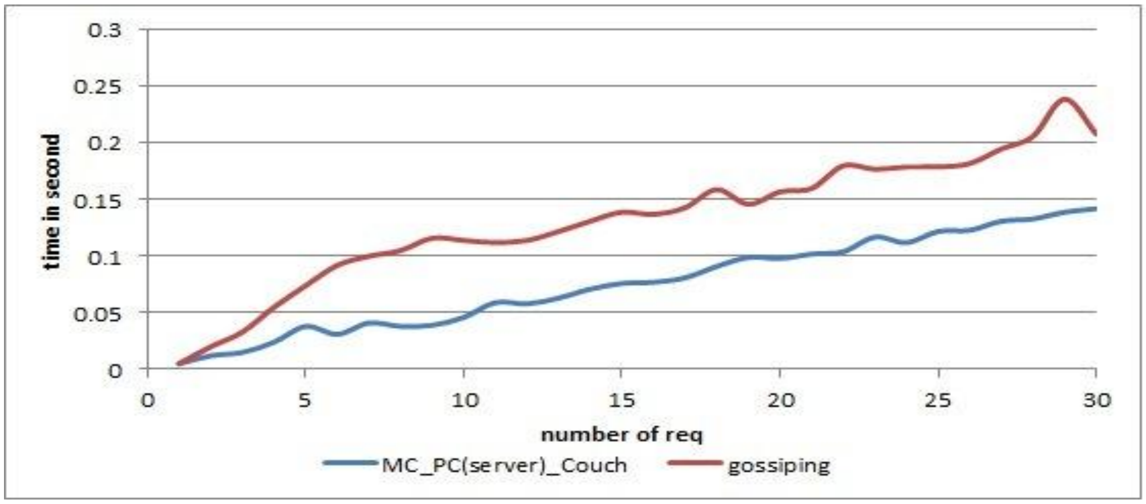


Figure 6-15. Comparison of maximum response times for two different proxies

Figure 6-14 and figure 6-15 show the comparison of minimum and maximum response times for two different proxies where response times are minimal as the number of requests is changed from 1 to 30 when proxy server A (presented by the blue line) is used over proxy server B (presented by the red line).

Table 6-6 shows the result of the three setups based on the average required response time in seconds for the services as the number of requests is changed from 1 to 30 using the proxy server A and proxy server B respectively for the BlackBerry 9550 Simulator. Table 6-7 shows the result of the three setups based on the average required response time in seconds for the services as the number of requests is changed from 1 to 150 using the proxy server A for the BlackBerry Playbook device.

Table 6-6. Average request-response duration using the BlackBerry 9550 Simulator

Experiment Setup	Setup A (no cache and no proxy server)	Setup B (client cache and no proxy server)	Setup C (proxy server A and cache)	Setup C (proxy server B and cache)
Average request-response time for sending 30 requests (s)	0.117367	0.119633	0.070607	0.121107

Table 6-7. Average request-response duration using the BlackBerry Playbook device

Experiment Setup	Setup A (no cache and no proxy server)	Setup B (client cache and no proxy server)	Setup C (proxy server A and cache)
Average request-response time for sending 150 requests (s)	0.235608	0.222295	0.026805

The results from table 6-6 and table 6-7 show that Setup C decreases the average round-trip time than the Setup A and the Setup B under the same conditions when proxy server A is used. On the other hand, Setup C takes longer duration for the average round-trip time than the Setup A and the Setup B under the same conditions when proxy server B is used which is shown in table 6-6.

Figure 6-16(a) shows the average required time in seconds for the services as the number of requests is changed from 1 to 30 for the 3 different models using the BlackBerry 9550 Simulator as the mobile client. Figure 6-16(b) shows the average required time in seconds for the services as the number of requests is changed from 1 to 150 for the 3 different models using the BlackBerry Playbook tablet device as the mobile client. Setup A (no cache and no proxy server), Setup B (cache in mobile client) and Setup C (proxy server and cache) for experiment 1 are represented in the graphs by the blue, red and green line respectively. Here I used proxy server A for the first part of the graph and proxy server B for the second part of the graph.

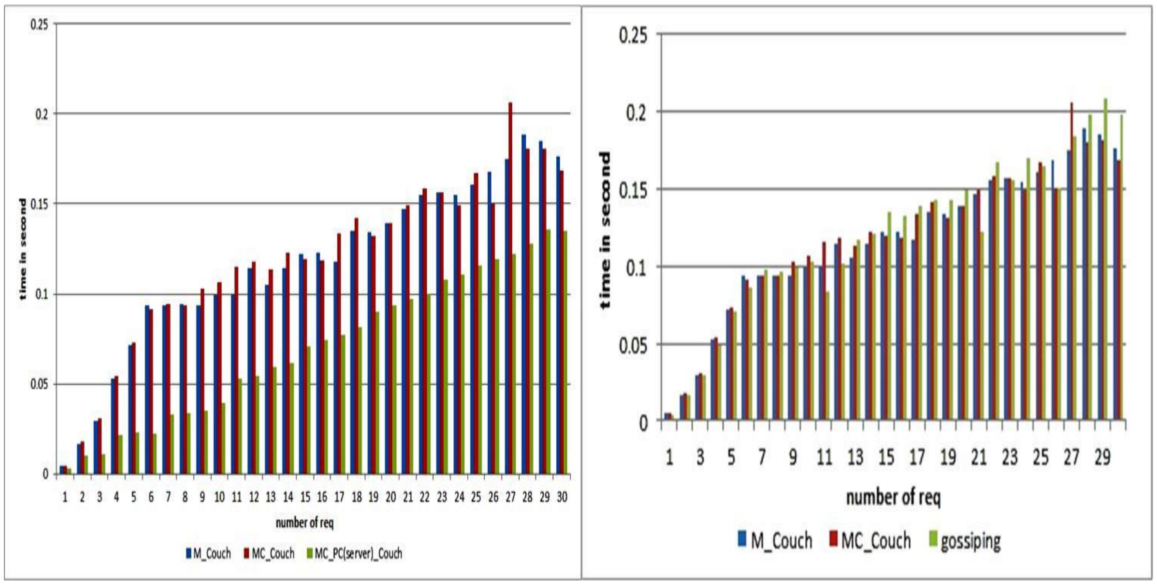


Figure 6-16(a). Average response times (using BlackBerry simulator)

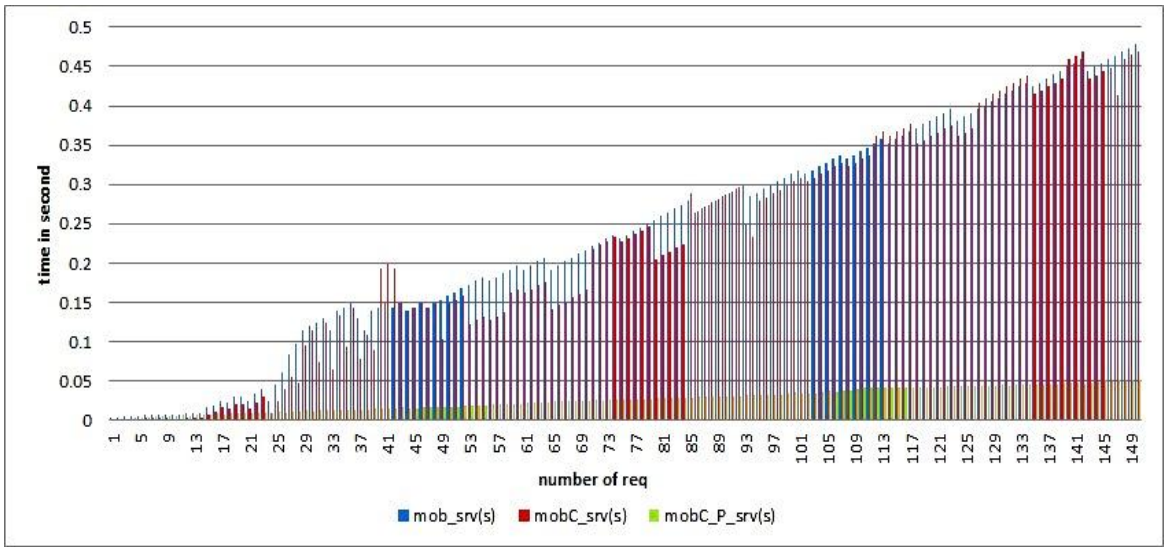


Figure 6-16(b). Average response times (using Playbook)

Figure 6-16(a) shows an increment of average required time as a function of the request number for the model A, B and model C (as we used closed loop for multiple GET requests) as the number of requests is changed from 1 to 30 when Proxy server A (the first part of the graph) is used. Though there is no notable performance increase happened when Proxy server B (the second part of the graph) is used for model C as the number of requests is changed from 1 to 30 in Figure 6-16(a).

In figure 6-16(b), average required time increases exponentially for model A and B while model C outperforms by showing a very linear line graph as the number of requests is changed from 1 to 150. The huge difference in model A and C; and model B and C are notably visible in this graph. The highest required response times for model A and model B are 0.47810029 seconds and 0.46910029 seconds respectively when the number of request is 150. On the other hand, the highest required response time for model C is 0.052 seconds for 150 requests which shows that model C takes shorter duration than model A and model B even with a large number of requests for the services.

Figure 6-17 shows the comparison of average response times for two different proxies. Proxy server A and Proxy server B (gossiping) are represented in the graphs by the blue and red line respectively.

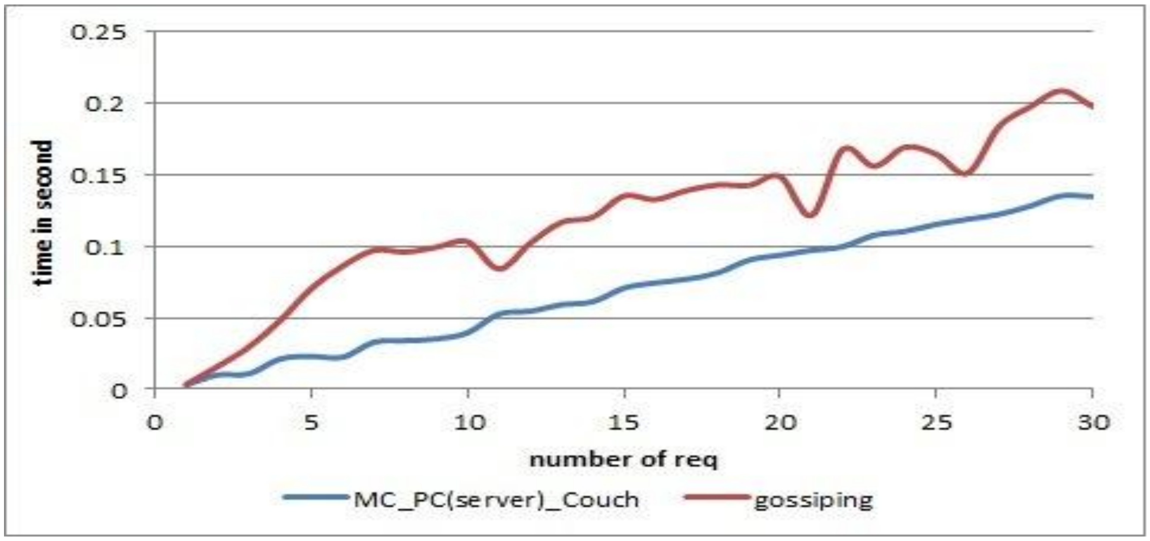


Figure 6-17. Comparison of average response times for two different proxies

In figure 6-17, the lowest required response times for proxy server A and proxy server B (gossiping) are 0.003 seconds and 0.0034 seconds respectively when the number of request is 1. The highest required response times for proxy server A and proxy server B (gossiping) are 0.1348 seconds and 0.1978 seconds respectively when the number of request is 30. The graph shows that proxy server B takes longer duration than proxy server A for the average required response times as the number of requests is changed from 1 to 30.

Experiment 3: Scalability Test

For experiment 3, for the scalability test, in the first and second setup, apache bench load generator generated 10000 workloads as the number of concurrent request is changed from 10 to

550 and sent it to the CouchDB REST server and Erlang proxy server respectively. The result of the round-trip test is presented in Table 6-8.

Table 6-8. Results comparing round- trip duration

Web Server	CouchDB REST server	Erlang proxy server (proxy server A)	Erlang proxy server (proxy server B)
Average request-response time for 10000 workloads (ms)	1.869181818	2.698545455	4.651927273

Figure 6-18(a) and figure 6-18(b) show the total required time in milliseconds for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads for the REST server and the Proxy server (using proxy server A and Proxy server B respectively).

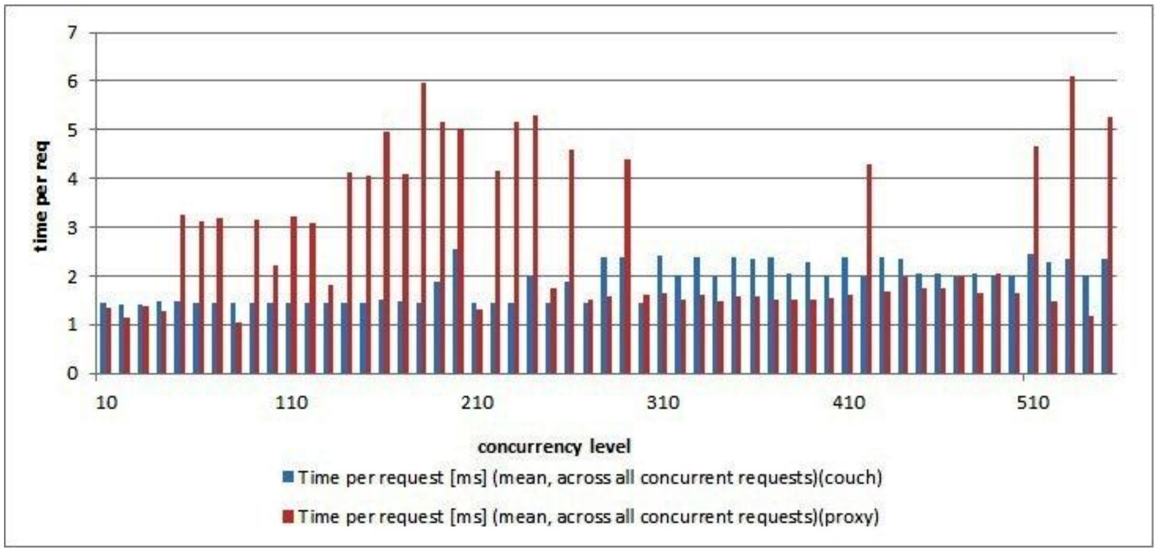


Figure 6-18(a). Time per request for proxy server A

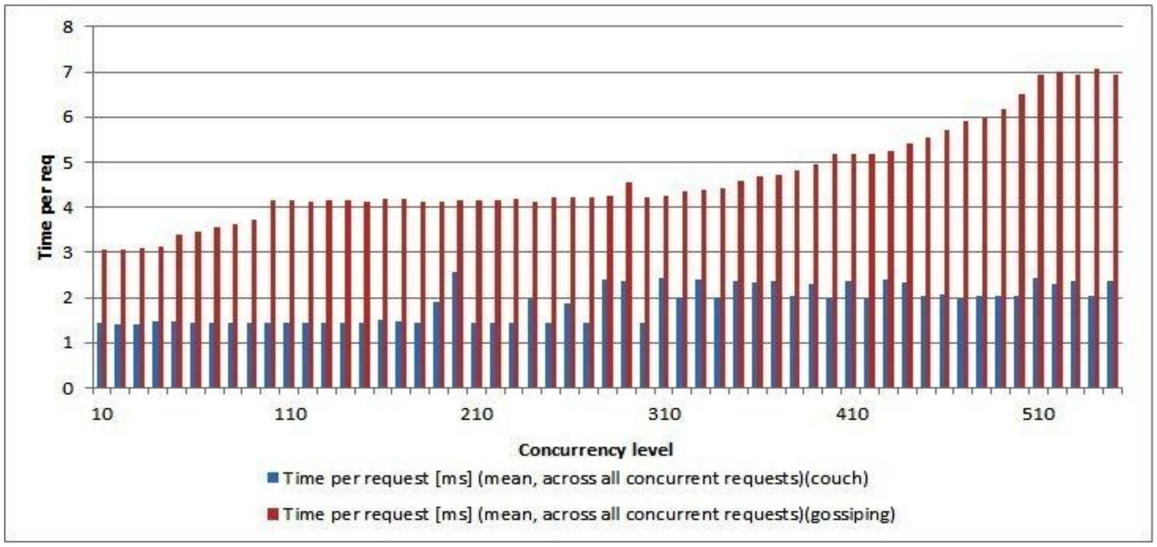


Figure 6-18(b). Time per request for proxy server B (using gossiping)

The result shows, in figure 6-18(a) and table 6-8, the proxy server A adds a little overhead (increased 44.37041% overhead for required time) than the REST Couch server. On the other hand, the proxy server B adds a lot of overhead (increased 148.8750547% overhead for required time) than the REST Couch server (in figure 6-18b and table 6-8). The reason of adding the overhead can be that the Proxy server A and Proxy server B are doing caching, cache validation, parsing and extracting data from the original result that is coming from the REST Couch server which takes time.

Figure 6-19 shows the comparison of total required time in milliseconds for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads for proxy server A and Proxy server B respectively.

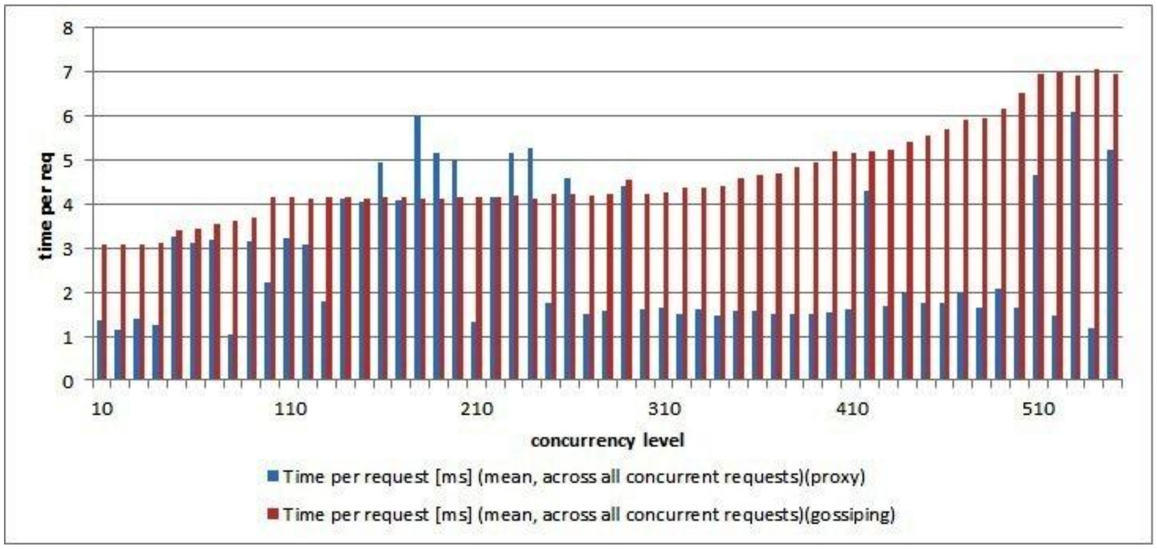


Figure 6-19. Comparison of time per request for two different proxies

The graph shows, in figure 6-19, Proxy server B adds a lot of overhead over Proxy server A (increased 72.38647083% overhead for required time) as the number of concurrent request is changed from 10 to 550 for 10000 workloads.

The analysis of the throughput calculation for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads for the REST server and the Proxy server is shown in table 6-9.

Table 6-9. Results comparing average throughput

Web Server	CouchDB REST server	Erlang proxy server (proxy server A)	Erlang proxy server (proxy server B)
Average throughput (s) for 10000 workloads	560.154	575.8693	219.0494545

The results from table 6-9 show that Erlang proxy server produced a percentage increase of 2.805532% throughput in comparison with the CouchDB REST server under the same conditions when proxy server A is used. On the contrary, Erlang proxy server produced a percentage decrease of 60.895% throughput in comparison with the CouchDB REST server under the same conditions when proxy server B is used.

Figure 6-20(a) and figure 6-20(b) show the throughput in seconds for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads for the REST server and the Proxy server (using proxy server A and Proxy server B respectively). Figure 6-21 shows the comparison of the throughput in seconds for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads for proxy server A and Proxy server B respectively.

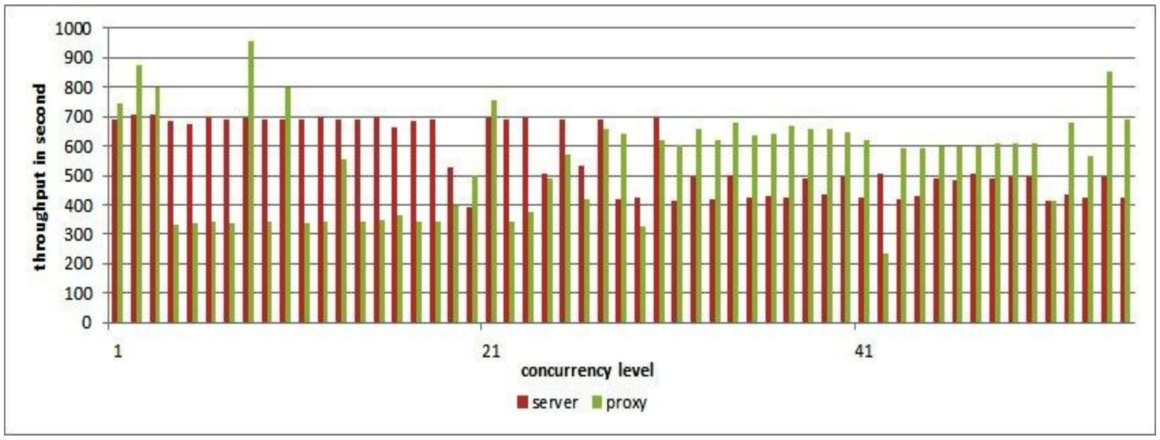


Figure 6-20(a). Throughput for proxy server A

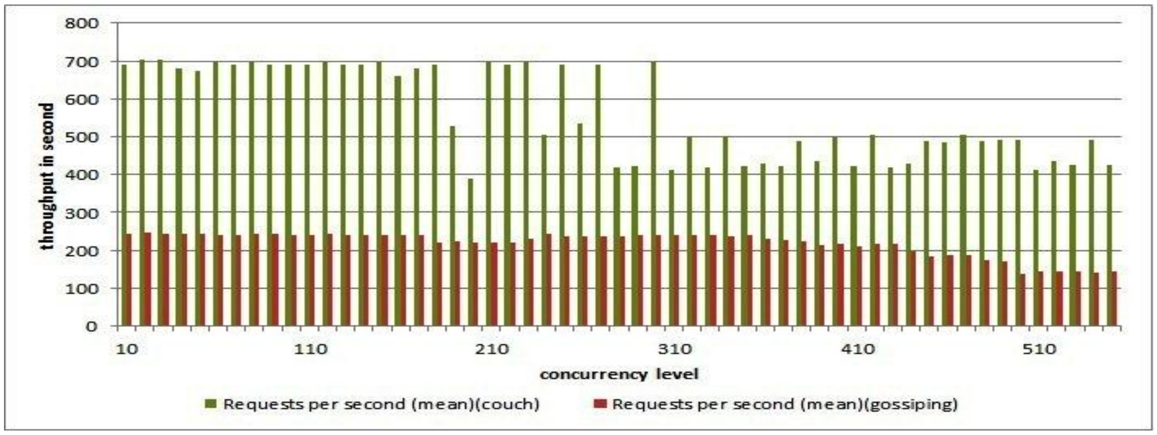


Figure 6-20(b). Throughput for proxy server B (using gossiping)

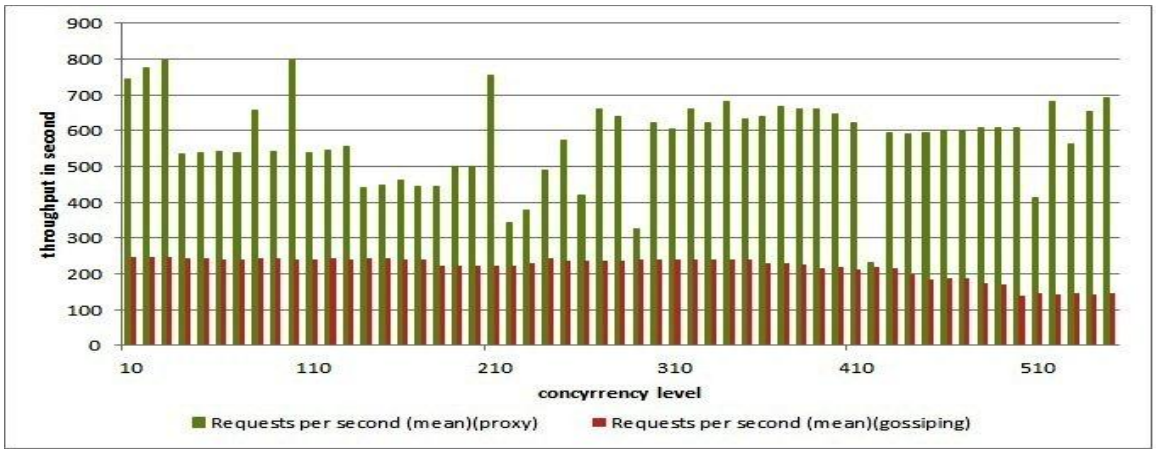


Figure 6-21. Comparison of throughput for two different proxies

In figure 6-20(a), the proxy server A has the better throughput than the REST server for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads. On the other hand, in figure 6-20(b), the proxy server B has a very low throughput than the REST server for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads. The graph in figure 6-21 also shows the vast difference of the throughput in seconds for proxy server A and proxy server B where proxy server A outperforms proxy server B.

6.5 Summary

The summary of the experiments are listed below based on the goals.

Goal 1. What is the maximum request/ response rate when the proxy server and the client communicate? This is to determine how fast the client and proxy server can efficiently communicate.

Experiment 1: To calculate the overhead, the systems were evaluated based on different numbers of concurrent clients that each executed 30 read requests (for the BlackBerry 9550

Simulator) and 150 read requests (for the BlackBerry Playbook device) to a RESTful web service respectively and plotted the throughput for three different setups. The result shows an increment in throughput for model C (Fig 6-11b), and a linear line for model C (Fig 6-11a). The graph shows a decrement in throughput for model A and B as number of requests increases from 1 to 30 in Fig 6-11a and exponential decrement for model A and B in Fig 6-11b. Also high throughput is achieved using model C over model A and B (table 6-2 and table 6-3).

Goal 2. How the state changes can be detected and propagated to the proxy server and the mobile client?

Experiment 2: To determine the delay between the state changes occurring and the client being notified, maximum, minimum and average required time is calculated based on different numbers of concurrent clients. Figure 6-12, Figure 6-13 and Figure 6-16(a) show an increase of required time as a function of the request number for the model A, B and model C (as we used closed loop for multiple GET requests) when Proxy server A is used. Though there is no notable performance increase happened when Proxy server B is used for model C (Figure 6-12, Figure 6-13 and Figure 6-16(a). On the other hand, for model A and B, average required time increases exponentially while model C outperforms by showing a very linear line graph in Figure 6-16(b). The huge difference in model A and C; and model B and C are notably visible in this graph. Also by comparing Proxy server A and Proxy server B for minimum, maximum and average response times in Figure 6-14, 6-15,6-17 and table 6-4, 6-5, 6-6 respectively, it is clear that Proxy server A performs far better than Proxy server B.

It is also very interesting to see that the performance is tremendously well when an actual device is used (in Figure 6-16(b)) for testing over a simulator (in Figure 6-16(a)) (table 6-6 and table 6-7).

Goal 3. How multiple devices of a single user can be supported by the cache efficiently?

Experiment 3: For the scalability test, in the first and second setup, apache bench load generator generated 10000 workloads as the number of concurrent request is changed from 10 to 550 and sent it to the actual CouchDB REST server and Erlang proxy server respectively. Figure 6-18(a) shows the total required time in milliseconds and Figure 6-20(a) shows the throughput in seconds for the services as the number of concurrent request is changed from 10 to 550 for 100000 workloads for the REST server and the Proxy server A. In both figure 6-18(a) and figure 6-20(a), we see that the proxy server A adds a little overhead (44.37041% overhead for required time) than the REST Couch server without optimizing the overall performance. Figure 6-18(b) shows the total required time in milliseconds and Figure 6-20(b) shows the throughput in seconds for the services as the number of concurrent request is changed from 10 to 550 for 100000 workloads for the REST server and the Proxy server B. In both figure 6-18(b) and figure 6-20(b), we see that the proxy server B adds a lot of overhead (148.8750547% overhead for required time) than the REST Couch server and does not advance the overall performance. The reason for the performance of Couch server is because CouchDB database also runs on Erlang Open Telecom Platform (OTP) [13] framework. The Proxy server A and Proxy server B add overhead with caching, cache validation, parsing and extracting data from the original result. However, Proxy server A performs better handling concurrency and number of requests which is visible in figure 6-20(a) where the proxy server has the better throughput graph.

While comparing Proxy server A and Proxy server B in figure 6-19 for total required time in milliseconds for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads, Proxy server A performs better that Proxy server B. Here Proxy server B adds a lot of overhead over Proxy server A (72.38647083% overhead for required time). Figure

6-21 shows the comparison of the throughput in seconds for the services as the number of concurrent request is changed from 10 to 550 for 10000 workloads for proxy server A and Proxy server B respectively where again Proxy server A outperforms Proxy server B.

Also, during the testing in experiment 3 for figure 6-18(a), 6-18(b) and figure 6-20(a), 6-20(b) it is observed that Erlang proxies respond successfully to all the concurrent HTTP requests ranging from 10 to 550 for 10000 workloads. This proves that the proxy server is reliable and high availability of the data while hosted in the cloud environment.

Also we see some spikes in the graphs which are probably caused by network traffic, simulator issue, device issue or other interference.

CHAPTER 7
SUMMARY AND CONTRIBUTION

7.1 Summary

This work puts forward a cloud-hosted proxy server approach as an effective way to support embedded browser apps on smartphones; and shows that RESTful Web services can be consumed by these resource limited mobile devices. The work proposes a caching technique on the proxy server to deal with the mobile network ecosystem challenges such as the sporadic (or unpredictable) network loss, high and intolerable latency, and limited availability of wireless bandwidth. The caching component is also extended to the mobile device in order to increase the availability of the RESTful Web resources. The distribution of the cache across the participating platforms is dubbed “dual caching;” a concept adapted from Liu et al. [29]. This research shows that caching and proxy server based techniques can be a solution for: addressing bandwidth limitations, minimizing latency to an acceptable level, recover from network loss, and managing the limited features (e.g. storage) of the mobile device.

Moreover, a prototypic mobile browser-based application called GradeView app is implemented based on the designed architecture. The app is developed using the embedded browser environment for the BlackBerry platform called WebWorks Blackberry and the Android platform called WebView. Both environments support code swapping so the entire code set is a single JavaScript base code.

The purpose of the GradeView application is to enable students to access their course specific grading records which are hosted on a REST Web server using their mobile devices. The use of the embedded browser framework made it possible to deploy a single code to different mobile platforms. I therefore, deployed and tested the GradeView app on various consumer device platforms such as the BlackBerry mobile, Android powered tablet and BlackBerry

Playbook tablet. The mobile client has its own JavaScript based caching component to support network loss, connectivity issue and bandwidth problems. The adoption of the REST WS standard is to manage the limited storage and processor of the mobile since REST traffic is lightweight and semantically easier to manipulate.

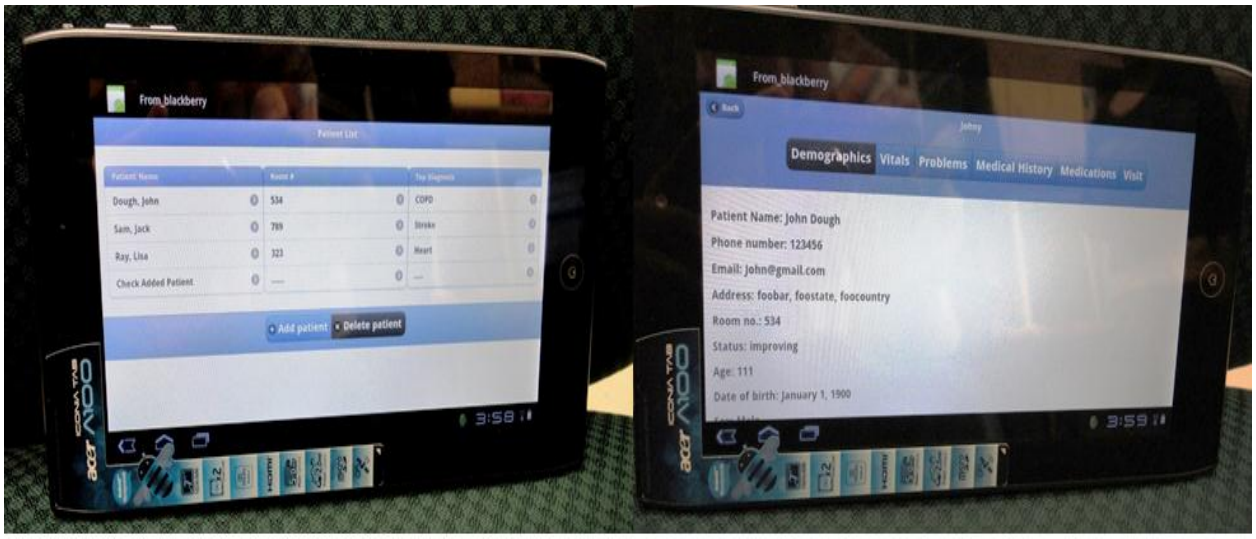
The proxy server was implemented using Erlang for providing high scalability and concurrency. Two different Erlang proxy servers were implemented and evaluated to see which one performs better and faster for state propagation to the mobile client. The first proxy is built based on the "gossiping" technique whereby the proxy checks from the REST server whether there is an update without the knowledge of the mobile client; and the second proxy adopts the general caching technique whereby the proxy stores replicas of all successful transactions between the mobile and the REST server. It is seen that the gossiping technique for the proxy server is responsible for decreasing the performance when the concurrency level increases. On the other hand, the proxy server with general cache validation policy performed significantly well and minimizes the communication latency. An environment identical to the Amazon EC2 cloud environment was used for hosting separately the REST Web server and the proxy server in order to ensure reliability and high infrastructure availability.

The results of the experiments prove that the use of caching and proxy server increase the performance significantly while consuming Web services over the wireless network. Thus, the use of a proxy server and proxy server caching can be more suitable option with client side caching while dealing with these resource poor mobile clients with RESTful Web services.

7.2 Implications

In this work, I built and evaluated a mobile application called GradeView app using embedded browser framework that can help students to access their grades for different courses using their mobile devices. For accessing a grade of a particular grade, the student will have to provide the course Id using the GradeView app in the mobile device and will see the corresponding course name and course grade displayed on the screen. The result of the experiments conducted using the app in a controlled lab environment are very positive and proved that caching and a cloud hosted proxy server can be a solution for addressing network loss, latency, limited resource and bandwidth problem while students are accessing their grades using their mobile devices.

This work can also be extended to the E-health field where patients and medical professionals can be aided to access information via mobile devices and reducing the waiting time as latency becomes vital in the medical domain. The framework can be used to build an application that will enable patients to access information, make appointments and to interact with health professionals via mobile devices. Similarly, allow the health care professionals to interact and push information to the patients. The caching and a cloud hosted proxy server can be a solution for handing network loss and latency thus making it easier to access information for patients and health professionals using their mobile devices. The E-health concept is evidenced in another study called SOPHRA [31]. The joint E-health project with the Geriatrics Ward of the City Hospital in Saskatoon Canada aims to aid the health professionals to access patient records using their mobile tablet devices, using the embedded browser framework. The SOPHRA framework used CSS, HTML5, and JavaScript for building the mobile client application [31]. For the implementation of the proxy server, the Erlang programming language is used to achieve high concurrency and scalability. Figure 7-1 shows the screenshot for the SOPHRA app.



Patient Name	Room #	Top Diagnosis
Dough, John	534	COPD
Sam, Jack	789	Stroke
Ray, Lisa	323	Heart
Check Added Patient	—	—

Patient Name: John Dough

Phone number: 123456

Email: John@gmail.com

Address: foobar, foostate, foocountry

Room no.: 534

Status: improving

Age: 111

Date of birth: January 1, 1900

Figure 7-1. SOPHRA app on Android tablet device [31]

7.3 Contribution

The contribution and the findings of our work are summarized below.

- A prototype implementation of a framework that enforces low latency in a distributed mobile environment.
- The mobile client is able to consume the RESTful WS through our proprietary proxy server.
- Offline RESTful resource accessibility on mobile devices in the case of temporal and short-lived disconnections.
- Embedded browser framework implementation for the mobile client side enhances the deployment of the framework for various mobile platforms.
- Proxy server can be used to as an effective way to get the Web services (resources) state propagation from the REST Web server.

- Client and proxy side caching can be used as an effective way to deal with the wireless network induced problems.
- The cache can be used for multiple devices as we tested it on different platforms.
- The Erlang proxy server can give good support to the mobile client.
- The proxy server and the RESTful Web server can be hosted on other IaaS-based clusters such as Amazon EC2 cloud environment.

CHAPTER 8
LIMITATIONS AND FUTURE WORKS

8.1 Distributed Proxy

This work presents the adoption of a cloud-hosted proxy server to support resource poor devices such as smartphones and tablets to consume RESTful Web services with less processing effort. In this work, only one proxy server is considered to sit between the mobile clients and the RESTful Web server to deal with high communication latency. The reason for the employment of a single proxy server is to minimize the data inconsistency that may be introduced when multi-proxy server approach is adopted in the mobile distributed system. However, as systems continue to grow in order to meet increasing processing and storage demands, it will be impractical to maintain a single proxy for an enterprise system deployment. Hence, the future work will investigate how multiple proxy servers can be implemented and maintained to increase the performance of our system.

8.2 Caching Policies

In this work, I used session storage [61] policy for storing resources in the mobile device cache, which is a browser dependent Web storage technique. I implemented the session storage mechanism in our GradeView app for maintaining the data in the mobile device because mobile devices are resource limited. Therefore, the saved data in the mobile cache is deleted as soon as the application is turned off. In my future work, I would like to explore the persistent storage or local storage [61] policy which allows storing data even after the browser is closed. The persistent data storage approach will enable the support for business continuity since users can have access to the data even when network connectivity is loss. The challenge that the persistent data storage on the mobile may introduce is the case of distributed stale cache. This situation will be investigated and data consistency policies will be proposed.

The future outlook on caching will also focus on the analysis of caching techniques based on context, metadata and so on.

8.3 Security Issues

Currently, security issues of the data on the mobile device are not considered. Since the current form of storage on the mobile follows the session storage technology, we expect the data to be erased the moment the application closes. However, as the future work is going to explore the local storage mechanism and keeping the cache data in the mobile device to handle network loss and latency problem, I would like to conduct a study to ensure the cached data is secure on the mobile device. Therefore, I would explore the authentication techniques for the mobile clients in my future work so that the data can be secured even if the mobile device gets into a wrong hand.

8.4 The Impact of Multi-tier Caching

In this work, a centralized proxy server is employed between the mobile client and the REST Web server in the distributed system and the “Dual Caching Model” is implemented based on the framework proposed by Liu et al. [29]. However, if the centralized proxy server crashes for any reason, no mobile client(s) will be able to access data from the REST server as all the requests are going to the REST server from the mobile device through the proxy server. In my future work, I would like to investigate the impact of multi-tier application caching in scale free networks [35]. The approach will enable the replica Web services to be stored on multiple nodes which may be distributed across different virtual, physical, and application layers.

8.5 The Impact of Database Size

In this work, the database I worked with was a moderately small database in terms of the size. However, if a larger database is used, the system might behave differently as I am using cache in the mobile device built in JavaScript which might not have the necessary support ability. Therefore, in my future work, I would like to implement and evaluate my framework based on different databases in terms of their sizes and study the behavior of the result.

The database size analysis is also beneficial for the studies of “Big Data” policy characterization. The amount of data being produced daily by the enterprise world is reaching billions of Terabytes and the bigger challenge is these data is becoming unmanageable. Detail studies in the future on the Big Data phenomenon will aid us to formulate policies to reduce the data pollution.

LIST OF REFERENCES

[1] Adamczyk, P., Smith, P. H., Johnson, R. E., Hafiz, M. REST and Web Services: In Theory and In Practice

[2] Alarcon, R., Wilde, E., & Bellido, J. Hypermedia-driven RESTful Service Composition, Published in the ICSOC 2010 International Workshops PAASC, WESOA, SEE, and SC-LOG San Francisco, CA, USA

[3] Allavena, A., Demers, A., Hopcroft, E. J. 2005. Correctness of a gossip based membership protocol. In Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing (PODC '05). ACM, New York, NY, USA, 292-301. DOI=10.1145/1073814.1073871. http://doi.acm.org/10.1145/1073814.1073871

[4] Amazon Elastic Compute Cloud. Last accessed: July 15, 2012. http://aws.amazon.com/ec2/

[5] Apache CouchDB Project. Last accessed: July 15, 2012. http://couchdb.apache.org/

[6] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M. Above the Clouds: A Berkeley View of Cloud Computing, Technical Report UCB/EECS-2009-28, UCB, Feb. 2009.

[7] Barnatt, C. Explaining Cloud Computing [Online], 10th May 2009, Available: http://www.explainingcomputers.com./cloud.html.

[8] BlackBerry PlayBook Review. Last accessed: January 11, 2012. http://crackberry.com/blackberry-playbook-review

[9] Box, D. Code Name Indigo: A Guide to Developing and Running Connected Systems with Indigo, Last retrieved from http://msdn.microsoft.com/en-a/magazine/cc164026.aspx#S1 February 14, 2011

[10] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web services description language (wsdl) 1.1. W3C Recommendation, mar 2001.

[11] Christensen, J. H. Using RESTful web-services and cloud computing to create next generation mobile applications. Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, p 627-633, 2009, OOPSLA 2009 Companion - 24th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications, OOPSLA 2009, Orlando, Florida, USA.

[12] Endler, M., Rubinsztejn, H., C. A. da Rocha, R., Sacramento, V. Proxy-based Adaptation for Mobile Computing.

[13] Erlang Programming Language. Last accessed: July 30, 2012. http://www.erlang.org/

[14] Extensible Markup Language (XML) 1.0 (Fifth Edition). Last accessed: July 25, 2012. http://www.w3.org/TR/2008/REC-xml-20081126/

[15] Falaki, H., Lymberopoulos, D., Mahajan, R., Kandula, S., and Estrin, D., 2010. A first look at traffic on smartphones. In Proceedings of the 10th annual conference on Internet measurement (IMC '10). ACM, New York, NY, USA, 281-287. DOI=10.1145/1879141.1879176 http://doi.acm.org/10.1145/1879141.1879176

[16] Fielding, R. Architectural Styles and the Design of Network-based Software Architectures, University of California, 2000.

[17] Fowler, M. Richardson Maturity Model: steps toward the glory of REST. March 2010 http://martinfowler.com/articles/richardsonMaturityModel.html

[18] Frangiadakis, N., and Roussopoulos, N. Caching in Mobile Environments: A New Analysis and the Mobile-Cache System, Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium.

[19] Fremantle, P., Weerawarana, S., Khalaf, R. ENTERPRISE SERVICES, Published in the COMMUNICATIONS OF THE ACM, vol.45,no.10, pp. 77-81, Oct. 2002.

[20] Gartner releases phone market share report for 2011, 15 February, 2012 http://www.gsmarena.com/gartner_releases_phone_market_share_report_for_2011-news-3832.php

[21] Gartner: "Market Share Analysis: Mobile Devices, Worldwide, 4Q10 and 2010", by R. Cozza et al. 08 February 2011.

[22] Gitzenis, S. Bambos, N. Joint Transmitter Power Control and Mobile Cache Management in Wireless Computing, Mobile Computing, IEEE Transactions on page(s): 498 - 512 , Volume: 7 Issue: 4, April 2008

[23] Gokhale, A., Kumar, B., Sahuguet, A. Reinventing the Wheel? CORBA vs. Web Services, Published in the Eleventh International World Wide Web (WWW2002), 2002.

[24] Hadley, M. J. Web Application Description Language (WADL), Sun Microsystems Inc: February 2, 2009.

[25] Hypertext Transfer Protocol – HTTP/1.1, RFC 2616: Last accessed: July 30, 2012. http://www.w3.org/Protocols/rfc2616/rfc2616.html

[26] Jamal, S., Deters, R. Using a Cloud-Hosted Proxy to support Mobile Consumers of RESTful Services. The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011). Procedia Computer Science, Volume 5, 2011, Pages 625-632, ISSN 1877-0509, 10.1016/j.procs.2011.07.081. (http://www.sciencedirect.com/science/article/pii/S1877050911004066)

[27] Lee, W., Lee, C., M., Lee, J., W., and Sohn., J., 2009. ROA based web service provisioning methodology for Telco and its implementation. In Proceedings of the 12th Asia-Pacific network operations and management conference on Management enabling the future internet for changing business and new computing services (APNOMS'09). Hong, C., Tonouchi, T., Ma, Y., and Chao, C., (Eds.). Springer-Verlag, Berlin, Heidelberg, 511-514. http://www.springerlink.com/content/k44131001v295679/

[28] Liu, D., and Deters, R. Management of service-oriented systems, Service Oriented Computing and Applications. vol. 2, pp. 51-64, 2008.

[29] Liu, X. and Deters, R. An efficient dual caching strategy for web service-enabled PDAs, SAC '07: Proceedings of the 2007 ACM symposium on Applied computing, New York, NY, USA: ACM, 2007, pp. 788–794.

[30] Lomotey, R. K. (2012). ENABLING MOBILE DEVICES TO HOST CONSUMERS AND PROVIDERS OF RESTFUL WEB SERVICES, M.Sc. Thesis Submitted to the College of Graduate Studies and Research, Department of Computer Science, University of Saskatchewan, Saskatoon, Canada. March, 2012.

[31] Lomotey, R. K., Jamal, S., Deters, R.: SOPHRA: A Mobile Web Services Hosting Infrastructure in mHealth. Proceedings: IEEE MS 2012, 1st International Conference on Mobile Services, Honolulu, Hawaii, USA, June 2012.

[32] MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., and Metz, R. Reference model for service oriented architecture 1.0. Technical report, OASIS, October 2006.

[33] Mahmoud, Q. H. Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). April 2005: http://www.oracle.com/technetwork/articles/javase/soa-142870.html

[34] Mobile Device. Last accessed: January 24, 2012.http://technologygear.net/tag/mobile

[35] Oliver, H., Michael, S. Scale-free Networks: The impact of fat tailed degree distribution on diffusion and communication processe. WIRTSCHAFTSINFORMATIK. Publisher: Gabler Verlag, Volume 48, Issue 4, 2006, Pages 267-275, ISSN 1877-0509, 10.1016/j.procs.2011.07.081. http://dx.doi.org/10.1007/s11576-006-0058-2

[36] Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. An Early Performance Analysis of Cloud Computing Services for Scientific Computing, Dec. 2008.

[37] Overdick, H. The Resource-Oriented Architecture, services, pp.340-347, 2007 IEEE Congress on Services (Services 2007), 2007

[38] Pautasso, C., Zimmermann, O., & Leymann, F. (2008). Restful web services vs. "big"' web services. Proceeding of the 17th international conference on World Wide Web WWW 08, 08, 805. ACM Press. doi: 10.1145/1367497.1367606.

[39] Proxy server. Last accessed: January 03, 2012. http://en.wikipedia.org/wiki/Proxy_server

[40] Qian, L., Luo, Z., Du, Y., and Guo, L., 2009. Cloud Computing: An Overview. In Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09), Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong (Eds.). Springer-Verlag, Berlin, Heidelberg, 626-631. DOI=10.1007/978-3-642-10665-1_63 http://dx.doi.org/10.1007/978-3-642-10665-1_63 L.M.

[41] Ren, Q., and Dunham, M. H., 2000. Using semantic caching to manage location dependent data in mobile computing. In Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom '00). ACM, New York, NY, USA, 210-221. DOI=10.1145/345910.345948 http://doi.acm.org/10.1145/345910.345948

[42] Schmidt, E. Conversation with Eric Schmidt hosted by Danny Sullivan. In: Search Engine Strategies Conference (August 2006)

[43] Selonen, P., Belimpasakis, P., You, Y. Experiences in Building a RESTful Mixed Reality Web Service Platform. Web Engineering. Proceedings 10th International Conference, ICWE 2010, p 400-14, 2010

[44] Seminar Reports On Cloud Computing. Last accessed: July 25, 2012. http://engineeringprojectsreports.blogspot.ca/2012/06/seminar-reports-on-cloud-computing.html

[45] Service Oriented Architecture. Last accessed: January 30, 2012. http://en.wikipedia.org/wiki/Service-oriented_architecture

[46] Simple Object Access Protocol (SOAP) 1.1. Last accessed: July 30, 2012. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[47] Stirbu. V., 2010. A RESTful architecture for adaptive and multi-device application sharing. In Proceedings of the First International Workshop on RESTful Design (WS-REST '10). Pautasso, C., Wilde, E., and Marinos, A., (Eds.). ACM, New York, NY, USA, 62-65. DOI=10.1145/1798354.1798388 http://doi.acm.org/10.1145/1798354.1798388

[48] Sullivan, D. (October 1, 2002), Death Of A Meta Tag, Last accessed: June 03, 2012. http://searchenginewatch.com/article/2066825/Death-Of-A-Meta-Tag

[49] Takase, T., Makino, S., Kawanaka, S., Ueno, K., Ferris, C., and Ryman, A. Definition Languages for RESTful Web Services: WADL vs. WSDL 2.0.

[50] Top Cloud Service Providers; Amazon, Google, IBM [Online], 24th October 2009, Available: http://www.mrwebmarketing.com/web-news/top-cloud-service-providers-amazon-google-and-ibm.

[51] Using ETags to Reduce Bandwith & Workload with Spring & Hibernate. Last accessed: June 7, 2012. http://www.infoq.com/articles/etags

[52] Vaquero, L. Rodero-Merino, L., Caceres, J., and Lindner, M. A break in the clouds: towards a cloud definition, SIGCOMM Comput. Commun. Rev., vol. 39, 2009, pp. 50–55.

[53] Wang, Q. Mobile Cloud Computing, A Thesis Submitted to the College of Graduate Studies and Research, Department of Computer Science, University of Saskatchewan, Saskatoon, Canada. January, 2011

[54] Wang, Z., Das, S., K., Che, H., and Kumar, M., 2004. A Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments. IEEE Trans. Parallel Distrib. Syst. 15, 11 (November 2004), 983-995. DOI=10.1109/TPDS.2004.60 http://dx.doi.org/10.1109/TPDS.2004.60

[55] Wang, Z., Kumar, M., Das, S. K., and Shen, H., 2003. Investigation of Cache Maintenance Strategies for Multi-cell Environments. In Proceedings of the 4th International Conference on Mobile Data Management (MDM '03), Ming-Syan Chen, Panos K. Chrysanthis, Morris Sloman, and Arkady B. Zaslavsky (Eds.). Springer-Verlag, London, UK, UK, 29-44.

[56] Web Application Description Language, 2009. Last retrieved from http://www.w3.org/Submission/wadl/, July 20, 2011.

[57] Web Application Description Language. Last accessed: January 30, 2012. http://www.w3.org/Submission/wadl/

[58] Web Services Architecture, 2004. Last retrieved from http://www.w3.org/TR/ws-arch/ February 10, 2011

[59] Web Services Essentials, Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, o'reilly. http://oreilly.com/catalog/webservess/chapter/ch06.html

[60] Web Services Glossary. Last accessed: January 29, 2012. http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

[61] Web Storage. Last accessed: April 24, 2012. http://www.w3.org/TR/webstorage/

[62] Webber, J., Parastatidis, S., Robinson, I. REST in Practice: Hypermedia and Systems Architecture.

[63] Weerawarana, S., Curbera, F., Leymann, F., Storey, T., & Ferguson, D. F. Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More, Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[64] Working of Proxy Server. Last accessed: January 7, 2012. http://en.wikibooks.org/wiki/Communication_Networks/HTTP_Protocol