A web based Publish-Subscribe framework for Mobile Computing

Cosmina Ivan
Department of Computer Science
Technical University of Cluj Napoca
Cluj, Romania

***Abstract*—The growing popularity of mobile devices is permanently changing the Internet user's computing experience. Smartphones and tablets begin to replace the desktop as the primary means of interacting with various information technology and web resources. While mobile devices facilitate in consuming web resources in the form of web services, the growing demand for consuming services on mobile device is introducing a complex ecosystem in the mobile environment. This research addresses the communication challenges involved in mobile distributed networks and proposes an *event-driven communication* approach for information dissemination. This research investigates different communication techniques such as polling, long-polling and server-side push as client-server interaction mechanisms and the latest web technologies standard *WebSocket,* as communication protocol within a Publish/Subscribe paradigm. Finally, this paper introduces and evaluates the proposed framework, that is a hybrid approach of WebSocket and event-based publish/subscribe for operating in mobile environments.**

Keywords—mobile computing; Websockets; publish-subscribe; REST

I. Introduction

In recent years, the growth of mobile devices such as smartphone and tablets has led to an extensive use of mobile applications in almost every sector of our life. The Gartner research [1] forecast 2011 states, that the download of mobile apps worldwide had increased by 117 percent from 2010 to 2011 and forecasts an astounding 185 billion downloads from mobile app store by 2014, since the first launch in 2008. The capabilities of these devices in doing more than just making calls as well as sending and receiving text messages has increased the demand for mobile applications in the enterprise, as it becomes possible for enterprises to extend their services to the fingertips of numerous consumers.

Generally, these mobile applications consume data as Web services from a remote server- based architecture, which is the backbone of most information systems. Today's information society is built upon collaborative platforms which gathers and shares information across distributed networks, so the backbone of these information systems consists of multiple disparate system applications. The growing demand of consumers in accessing services is causing these systems to expand and some of these services can be hosted in the cloud computing environment, in order to ensure availability, reliability and scalability in service consumption. Cloud computing is the era where IT services are outsourced from providers over the internet on pay-according-to-use policy [2]. With the growing demand of consumer web services and the expansion of systems that forms a gigantic distributed heterogeneous infrastructure, there is an acute need for frameworks that can reliably operate in the mobile environment.

The remainder of the paper is organized as follows. Section 2 reviews some of the key points that this study explored and the existing research works within the identified problem domain. Section 3 presents the proposed framework design in addressing the research goals and challenges. Section 4 describes the implementation details of the architecture followed by the experiments designed to verify the framework in accordance with the research goals. Finally, section 5 concludes the thesis with the contributions of this research.

II. Problem Specification and Literature Review

While distributing the system applications provides more flexibility and scalability, it often results into a growing system complexity during services consumption in a mobile environment. One of the major challenges in today's enterprise solution is to ensure integration among these disparate and distributed system applications which are often connected to legacy systems. In addition to that, mobile devices are becoming an integral part of the growing digital ecosystem and the primary means of accessing IT services. The major challenges while disseminating data over a wireless connection in a mobile environment are as follows: unreliable network connection**,** higher degree of network latency, limited network bandwidth.

This introduces more challenges to the system when synchronizing the information flow between mobile clients and the distributed system backend.

A. Problem specification

In addressing the above mentioned challenges in mobile digital ecosystems, this research looks into *developing a framework* for disseminating data over wireless networks and *proposes an architecture* that allows system components to independently propagate data (i.e. resource updates) and as they propagate, the *eventual consistency technique* is employed to synchronize the data . In this regard, this paper looks into the *Pub/Sub pattern* as a mechanism for propagating data close to real-time, moreover, the emergence Web 2.0 has greatly embraced the Restful web services [3] due to its web compliant API and lightweight solution for

resource's state management. Therefore, the proposed framework is a hybrid of REST-based and event-based Pub/Sub that deploys a combination of various client-server interaction modes, such as polling, long- polling and server-side pushing.

The main research goal in proposing such a framework for mobile devices is to integrate REST web services within Pub/Sub domain. In this respect, the research will look into different *Rest patterns* in disseminating data, chose the most suitable for an event-based Pub/Sub system and address the above mentioned challenges in wireless network. The secondary research goal is to reduce network latency, bandwidth usage and also synchronizing resource's state in the face in intermittent connection loss, in terms of the proposed implementation.

The remainder of the paper is organized as follows. Section 2 reviews some of the key points that this study explored and the existing research works within the identified problem domain. Section 3 presents the proposed framework design in addressing the research goals and challenges. Section 4 describes the implementation details of the architecture followed by the experiments in section 5 designed to verify the framework in accordance with the research goals. Finally, section 6 concludes the thesis with the contributions of this research.

B. Literature review

A communication model that helps in dealing with the information dissemination in a large scale mobile network is Pub/Sub paradigm [4]. In this Pub/Sub architecture, information providers as publishers disseminate information in the form of events and information consumers ,as subscribers register for events of their own interests. There can be an event broker acting as a middleware which helps in dispatching events to the respective subscribers.

Communication in Pub/Sub is inherently asynchronous and transparent in nature as both entities (information provider and subscriber) operate asynchronously through a dispatcher and disseminate state changes to all interested subscribers through one operation. In the basic model of a Pub/Sub system, both providers and subscribers are connected through a set of groups or channels through which subscribers are notified for the events of their interest. Upon receiving event notification, the publisher dispatches the event to the respective subscribers.

As subscribers are not interested in all the events that are published by the providers, there are various ways that the subscriber can specify interest for a specific event. These variations have led to different subscription models that are currently seen in Pub/Sub system environments. The most important subscription models are topic and content based schemas.

One of the first generation subscription schemes is the topic-based scheme. In this scheme, subscribers register for notification based on the topic or subject of the events corresponding to a particular group ,or a set of groups also known as a logical channel [5]. Users subscribed to a channel(s) will receive all published events of that channel. The topic-based scheme has been proposed as a solution in many industrial Pub/Sub environments, one of the most mentioned systems is CORBA notification service and DDS from OMG group, also among others, and TIB/RV, SCRIBE and Bayeux are some of the systems that implement topic-based scheme **[**5**]**.

The Pub/Sub paradigm is better understood in the domain of a *messaging system* and also known in the domain as Pub/Sub messaging system, and has the capability of managing messages in a similar way that a persistent database is managed by a database system. Messages are coordinated and integrated among the software components as software applications changes over time, and are transferred from one machine to another over the unreliable wireless network.

A more flexible but also complex paradigm in the Pub/Sub scheme is content-based subscription. It provides more flexibility to the subscriber by providing more control in subscribing an event based on the actual content of the event. It allows subscriber to impose set of constraints in the form of condition in forming a query on an event notification (also known as *filter*). Creating a notification using a filter provides subscribers with a more sophisticated way for subscribing events. However, this higher expressive capability in defining subscription on the other hand, can be an added challenge in implementing such a scheme, since matching publisher's events with subscriber become more complicated and the resource consumption becomes higher, inappropriate to our goals [5]. There are several examples of systems that implement content- based subscription scheme such as Siena, Jedi, and Rebeca [6].

The inherent limitations of wireless network makes the messaging system suitable to operate as it repeatedly tries to transmit message until it has been sent. The basic concepts in a messaging technology revolve around the key terms of message, channel and routing messages, and they will be extensively used. Transmitting data in sending messages back and forth has many advantages in a distributed application system. Some of the major advantages [7] are:

Asynchronous communication - in asynchronous communication a sender doesn't need to wait for the response to come in order to send the next request.

Throttling - a problem with messaging in Remote Procedure Calls (RPC) is that the receiver may crash due to the overhead of incoming messages. A messaging system has control on the number of requests to be sent to the receiver to process which saves the receiver from crashing.

Reliable communication - messaging system uses a store- and -forward style in providing a reliable delivery of messages.

C. Pub-sub in mobile environments

There are several papers that analyse the existing Pub/Sub model and his implementation mostly for the content-based subscription and suggests more enhanced approaches based on various optimisations. These approaches can be adapted into a mobile environment considering mobility issues of Pub/Sub system elements.

In [1] was proposed a middleware approach for a Pub/Sub implementation and its adaptation for a mobile environment. The authors explains how an event broker as a mediator can facilitate Pub/Sub communication in both centralized and decentralized mobile environments and proposes an algorithm for an optimized wireless network communication. The paper addresses the challenges of mobile networks in terms of network disconnection at any certain point and suggests the replication of users' subscription over multiple event brokers in order to improve the availability and reliability of the system in a mobile environment.

A scalable decentralized peer-based subscription approach implementation of Pub/Sub system has been proposed by authors of [8]. The study presents a topic-based deterministic information dissemination scheme that provides transparency for publisher and subscriber.

Another content-based Pub/Sub middleware approach has been proposed in [9]. The concept of mobility has been segregated into two parts – the physical mobility and the logical mobility. Depending on logical mobility, a new approach of 'location dependent subscription' using location-dependent filter has been introduced by author. In addition, the goal of [9] is to support mobile client applications in a decentralized Pub/Sub environment where clients are connected to one of the interconnected access points that serve as message routers in a distributed network. The paper implements a 'mobility support service' that provides this support to a mobile client by introducing independent mobility service proxies running at the access points of the Pub/Sub system.

A logical orientation scheme in subscription model also ensures a space optimized information Two key problems that arise in mobile applications in Pub/Sub system that have been addressed in [10] are namely scalability, in supporting large number of mobile clients and adapting to application topology as mobile components are subject to change their locations. TOPSS and JEDI are two examples of Pub/Sub systems that address scalability issue by implementing an efficient filtering mechanism at the event broker.

Although different implementations of mobile Pub/Sub systems have different prototypical and standard approaches, the common goal in all of these implementations is achieving an *efficient data dissemination strategy*. The objective of data dissemination is to transfer dynamic information (state) changes as a consequence of publishing new data and updating existing data from publishers to mobile consumers [11].

In today's heterogeneous networks that consist of Wi-Fi, 3G or 4G networks, most of the client consumers in Pub/Sub systems are smart phones and tablets, running native apps or mobile Web apps. From the developers perspective it is a controversial issue when it comes to developing apps for mobile devices. Native apps are developed solely for mobile devices which are accessible via specific device platform such as Android, Blackberry and iOS with a full access capability into the core device features. Mobile Web apps on the other hand provide the platform for single code based solution to be deployed on mobile devices with similar and more improved user experience as native apps. Thus , the mobile web app design reduces the cost of building and maintenance of mobile centric applications, and the mobile browser pattern has become the de facto standard for mobile applications since the Web is everywhere.

One key benefit of adopting mobile web methodology is the use of the latest HTML5 oriented web technology frameworks. Web frameworks such as Phone Gap and Sencha [12] support diverse mobile operating systems and allow mobile web developers to leverage their web technology skills in creating appealing applications. Moreover, these frameworks facilitate dynamic access capabilities to the device native features.

D. Web communication techniques

As a result, mobile web applications nowadays are gaining much popularity among the applications developers across several device platforms as well as in Pub/Sub system environment in disseminating information. Two of such strategies are: pull and push. In the pull approach, communication is initiated by information consumer whereas the push approach relies on information producer in initiating the communication [11]. Several web technologies are found to implement pull and push strategies. Three of such strategies expressed in their counterpart technologies, are conceptually known as *polling, long-polling and Web Sockets*. A real-time web application must receive up-to-date information. When the client browser (consumer) sends HTTP requests to the server (publisher) over a TCP connection, server acknowledges the request and issue a response back to the client.

Polling is one technique introduced in delivering real time information, in which, the client browser sends HTTP requests to the server at a regular time interval and every time the server receives a request, responds back to the client. This approach is suitable in a situation when the server update interval is known to the client so that the client can be synchronized to send request to the server based on the exact interval of message delivery. There is also a growing need for asynchronous communication in collaborative applications where multiple users interact real-time among themselves. To response to this need, the Ajax technique has been introduced which enables web browsers to fetch dynamic information from the server asynchronously using in-built JavaScript functionalities such as XMLHttpRequest. However, although Ajax solves the problem of collaborative communication, its intense communication with the server causes significant overhead especially when using the polling technique. As it is difficult to predict update interval of message dissemination in real-time application, polling data from the server with a long interval can make the communication slower whereas polling data with a short interval can result in many unnecessary HTTP requests with empty responses which causes lots of unnecessary HTTP responses.

Long-polling addresses the limitations of polling by avoiding sending request in an interval. In long-polling, as the browser initiates a HTTP connection with a server, the server maintains the connection persistently for a certain period of time and pushes the update message to the client whenever it becomes available. If the update is not available within the set

period of time, the server sends an empty response message as it times out and the connection is terminated. The browser then has to re-open another HTTP connection to send the next update request. In the asynchronous long-polling operation; the server can push update messages to the browser without the client prompting. However, performing long-polling in a groupware application where data is constantly updated will result in no improvement over the traditional polling technique as long-polling throttles the connection with lots of intermediate requests that consumes server resources [13].

Web Socket Technology. One of the latest web technology concepts introduced in the HTML5 standards as a new approach for the next generation web communication is Web Socket. It provides a *full-duplex bi-directional asynchronous* communication channel between web browser and web server applications over a single TCP socket per end point [14]. In addition, it has added the socket functionality to the browser to eliminate many problems of existing technologies. The complete Web Socket standard is the combination of the Web Socket API and the Web Socket protocol.

The Web Socket API is a draft specification standardized by W3C [14]. The API defines a communication interface between the web application and the browser [13]. The browser must expose the API to the web application so that when initiating a Web Socket connection the application invokes the following API to create a Web Socket object. Using the object, application then invokes the Web Socket API functions to open and close connection as well as send and receive messages. Current the browsers that support Web Socket standard are Firefox 6, Google Chrome 16, and Internet Explorer 10 [14]. *The Web Socket protocol* has been designed to improve the existing HTTP connection. Two primary tasks that this protocol performs are establishing connection through handshake and transferring data. The initial handshake starts with a HTTP protocol. In the browser request, the GET method indicates the end point of the connection. The Web Socket server uses values from headers sec-Web Socket-Key to calculate a hash value and send it to the client to prove that the handshake was received and sec-Web Socket-accept header field indicates whether or not the server accepts the connection.

Once the handshake between the client and the server is successfully established, the connection is ready for data transfer. In the Web Socket protocol, data is composed of sequence of frames which can be of type texts, interpreted as utf-8 text, binary data and control frame. Control frames are texts that are intended for signaling the connection for instance when the connection should be closed. Since the Web Socket protocol uses a HTTP compatible handshake, it can also use a HTTP port as well as an underlying TCP protocol for network communications. Several web-based systems are found nowadays are using the Web Socket API and the protocol as the key implementation tool. A web-based control application using Web Socket is proposed in [15] that shows how a Web Socket-based application can be built with just HTML5 without using any add-ons in the web browser. Another work by [16] integrates the Web Socket API into an existing framework to support distributed and agent-driven data mining in an enterprise environment. The work is similar to R- Web Socket except that it implements both the client and the server side interface for Web Socket API and the implementation uses Grizzy framework to provide scalability to the underlying infrastructure.

E. Application development patterns

A good architectural pattern in developing software applications can ensure a better performance for resource constraint mobile device. In talking about application design, we often encounter the term 'MVC' which is a short form of Model-View-Controller. An architectural design that is based on MVC produce a clear abstract framework in the system development process. This provides a clean separation between software components. An evolved version of MVC is MVP, stands for Model-View-Presenter that focuses on improving the presentation logic/UI logic. Unlike MVC, the *Presenter* component in MVP contains the user interface business logic of the *View*. Communication between *View* and *Presenter* thus happen through a view interface. As the UI logic of the *View* is dedicated to the *Presenter*, a direct request from *Presenter* to *View* becomes possible. *Presenter* can trigger the *View* updates without visiting though the *View* component. This is often considered as a reason in taking MVP pattern most suitable for web-based architecture. The separation of concern in presentation logic helps *Presenter* to ignore implementation details of the *View* and only concern on the method to invoke of the *View* interface. This feature of MVP provides a higher level of abstraction which made it a successor to MVC.

The traditional web application supports sequential flow of data where user had to fill a form and submit before showing the html content on the page. With the advent of AJAX, the modern UI of MVC/MVP supports *event-driven style of data flow*. As the stream of events arrives, the job of dispatcher is to determine the event type and pass it to the handler that can handle events of that type. In a client-server interaction, dispatcher and the event handlers may reside in the server side In that case, events from client's requests are queued up before transmitting them to the server to be processed. In event-driven, programs are like multiple individual modules that can be triggered based on the event types. The program is designed as a continuous loop that keeps listing for event and calls the event handler (also known as callbacks) that matches the event type.

F. Cloud computing

The foundation of cloud computing is seen as a remarkable way in consuming web services in resource poor of mobile device by offloading resource intensive computation and data storage outside the device into resource rich remote machines [18]. Computing in the cloud also provides scalable hosting of IT backend services. Several approaches have been proposed by myriads of research studies for the effectiveness of offloading techniques. Since the wireless signal may attenuate due to device mobility, these studies offer a notion of *dynamic offloading* that is said to be feasible in such network environment. [17] offers a cloud infrastructure that seamlessly offloads execution from mobile device to a replicate copy of mobile application software running in the virtual cloud server. This approach of migrating computation from a device

to a device replica gives mobile user an illusion of using powerful, feature rich device and also known as Clone Cloud. Similar approach is proposed by [16]. This study proposed to locate the cloud service software on a nearby resource-rich computer(s) called *cloudlets* that is well connected to the internet as well as to the mobile users. The approach of bringing the cloud virtual machine close to the mobile users is considered latency optimized in terms of latency and data transfer cost. In offloading mechanism, a fine grain offloading approach has shown in MAUI system [19] where instead of offloading on the whole application software, which methods to be executed remotely are decided in the runtime and thus saves energy and increase the battery life of mobile devices. Combining *cloud computing and RESTful Web services* provides a new paradigm of mobile computing. In his research specifies REST as a suitable architectural platform that lends itself well in consuming cloud Web services in resource constraint mobile device.

From the literature review, it can be concluded that the *channel based Pub/Sub* is an ideal model for a distributed system where applications are disparate and dispersed over the network. The space decoupling nature of Pub/Sub enabled mobile applications and the interacting parties who use these applications to be anonymous and independent from each other. Publisher can publish events at any time without blocking themselves and subscribers are notified asynchronously through a callback. Publisher doesn't hold any reference of subscriber which let the publisher to publish events even when the subscriber is disconnected. This decoupling in production and consumption explicitly removes dependencies among the interacting participants and increases the scalability.

The communication in Pub/Sub is asynchronous that well adapts with the distributed environment such as mobile environment. On the other hand, Web services have been a great solution in integrating distributed and disparate system applications. Due to clear semantics and uniform interface and its supportability for different message formats, REST Web Services has become the most suitable approach in consuming services in mobile environment. REST avoids the single access point in consuming services and thus increases the service scalability.

Reviewing the challenges in mobile distributed environment and the proposed solutions, this research attempts to address the following open issues;

- How can we build a RESTful Pub/Sub system in mobile environment?
- How much the system needs to comply with REST and Pub/Sub features to call it RESTful Pub/Sub?
- And because of operating in mobile environment, how can we ensure a system that is fault-tolerant and yet efficiently disseminate information?

In the rest of the paper we will try to respond to these questions in terms of a design solution, a prototype implementation and set of scenarios in order to make a realistic evaluation of our framework.

III. The Framework Design

This section looks into different REST patterns in event dissemination in accordance to the challenges mentioned in problem statement and then propose a framework that is adopted for mobile clients to consume RESTful Web Services within an event-based Pub/Sub domain. The proposed framework is designed in three main layers as shown in Figure1.

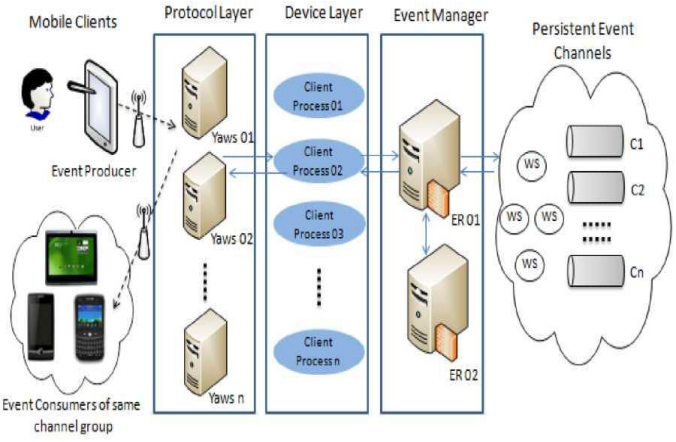

Fig. 1. The conceptual architecture

The front-end of the framework represents mobile clients who are publishers and/or subscribers of data at the Web Service (WS) channels. The backend of the framework contains Web servers as Protocol layer and Device layer, Event Manager and the cloud hosted Web Services channels. The Web servers and Event Manager act as a proxy layer between mobile clients and WS channels. Since *we adopt a Pub/Sub model*, data are disseminated in the form of events. Similarly, a mobile client that publishes events is known as the Event Producer (EP) and subscribers of these events are the Event Consumer (EC). However, an event consumer can be an event producer and vice versa. In this framework, *topic-based persistent event channels* were adopted. In topic-based persistent event channels, event producer publishes events to a specific channel topic and the event consumers show their interests for events by registering to a specific channel topic.

Event channels are collections of events represented by the event topic. In the Pub/Sub model, events are published using a single input channels that splits into multiple output channels to multicast the events to each subscriber. In the application-level, mobile client applications include User Interface (UI) layout, the business logic, and the model for managing a local storage. A stub component in the client model interacts with the skeleton of the server application. The persistent event channels are fronted with the Event Router component. , that takes the responsibility of multicasting events to the mobile subscribers. The client application includes a UI layout, the business logic and the local storage capability.

The client stub provides the functionalities of the backend server on the local device. On the contrary, the skeleton on the backend server describes the functionalities of the server application. The actual implementation of the skeleton is done

at the persistent event channel. Further, the Event Manager works as an intermediary between the skeleton and the persistent event channel. All message exchanges between the client device and the remote server takes place over the standard TCP/IP transaction layer.

TABLE I. PUB-SUB MAPPING TO REST SERVICES

Pub –Sub operation	REST model
Create Channel	POST/channel
Subscribe Channel	POST/channel/channel/topic/subscribe
Publish Events	POST/channel/channel_topic/publish
Read Events	GET/channel/channel_topic_eventMessages
Request for Updates	HEAD/channel/channel_topic
Unsubscribe Channel	DELETE/channel/channel_topic/unsubscribe

According to the Richardson's Maturity Model (RMM) [8], a RESTful dissemination of data can take four different patterns based on REST Web Service's maturity level also known as the glory of REST.

In the context of the proposed framework in this thesis, the patterns are hereby discussed as follows:

Pattern A: Using HTTP POST (Level 0) - Event-dissemination of this pattern follows level 0 of the RMM. In this pattern, services are exposed using one URI; and consumers can access the URI using a single HTTP POST method. This is similar to SOAP based WS where requests are sent to one URI and XML payloads are exchanged between the sender and receiver.

Pattern B: Using HTTP GET or POST (level 1) - Event dissemination of this pattern is based on level 1 of the RMM. In this pattern, a service is exposed as many logical resources with unique URIs contrary to single resource/service of level 0 (pattern A). A request is sent either using HTTP POST and/or HTTP GET. In this pattern, operations can be performed using HTTP POST. Sometimes HTTP GET is used in addition to HTTP POST. However, HTTP verbs do not strictly follow HTTP rules or REST constraints in this pattern.

Pattern C: Using HTTP CRUD Operations (level 2) - Services in this pattern host numerous URI-addressable resources. Unlike level 0 and 1 of the RMM, coordinating interactions in this pattern utilizes all the HTTP verbs (GET/retrieve, POST/create, PUT/update, DELETE/delete) in performing the CRUD operations. A response message in this communication utilizes the http status code

Pattern D: Using Hypermedia (level 3) - Pattern D is similar to pattern C in a way that it utilizes all the HTTP verbs in performing the CRUD operations except that it also utilizes the hypermedia element of the HTTP stack of the Web technology in the response message**.** Consuming services in a Pub/Sub framework can be challenging when complying with REST features described. This section describes how interactions can take place in REST-based manner in the proposed Pub/Sub based framework. Interactions between Web services and the service consumer are described in terms of major functionalities provided by the Pub/Sub service (as in Table 1).

A. The backend architecture

The backend server is responsible for hosting Pub/Sub Web Services. Web Services enables clients to create event channels (event groups) and publish events to the channel, subscribe to the channel(s) of their interests, be notified for resource updates of the channel and also unsubscribe from the channel. The system architecture takes a centralized topic based Pub/Sub model. The major functional components of the framework backend are shown in Figure 2, and further discussed.

Protocol and Device Layer. When an event is published in the event channel, it needs to be propagated as an update notification among respective subscribers. A published event is composed of event type, *etype*; published time, *etimestamp* and event messages, *emessage* (payload).

A published event is received by the Listener before it is transferred to the Event Manager (EM) process. It contains separate request handler for compatible transport mechanism.

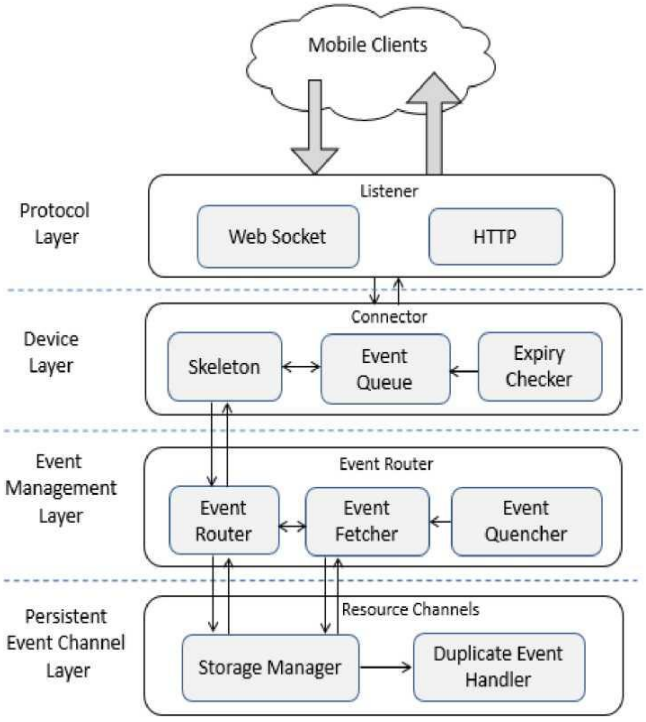

Fig. 2. Pub/Sub Backend Components

The expected transportation mechanism is the standard HTTP connection and/or Web Socket connection. Since mobile clients are using different types of device platforms, the embedded browser of native device application may not support either of this connection at any given time. To provide device transportation compatibility, a Listener process manages the request handlers for both HTTP and Web Socket. The device layer is responsible for redirecting client requests to the web services for appropriate operation execution using the connector process. This helps mobile consumers to maintain a presence at the proxy when they are disconnected

and thus resume the interaction with backend once the connection has been restored.The skeleton component of device layer provides the interface layer for Pub/Sub service, describing the functionalities that the service provides.

The Event Queue (EQ) component of the device layer buffers event update notifications received from Event Manager. It also handles duplicate event notifications to cope with network inconsistency. Event notifications are buffered in the queue until it has been propagated to the client device in FIFO style. An event is persistently removed from the queue once it is delivered to the consumer. Notifications in the Event Queue might become obsolete when event consumer is disconnected for relatively a long period of time. An event that is too old than the expected event longevity, need to be discarded from the event queue. The Expiry checker in the layer does a periodical checking in the event queue to ensure that no event notification in the queue is obsolete. Device layer also stores event data into the process storage based on their notification ID.

Event Manager (EM). The Event Manager is responsible to route event notifications to all the users who are subscribing to the channel group. Once an event is published to the persistent event channel, Event Manager invokes the Event Fetcher (EF) to fetch the list of all subscribed users of that channel. Consequently, the Event Router (ER) is invoked to actually send event notifications to the users from the subscription list. Dissemination of event updates takes a broadcast approach in delivering data to all currently active subscribers.

The Event Manager is also responsible to discard published events that arrives and does not match with the existing channel groups. An unmatched event is discarded when they are received at Event Manager. According to [Huang, Y., Molina, G., 2001], this approach is also known as event quenching. Discarding unmatched events considered to be advantageous as it does not require Event Manager or any of its replica (if any) to attempt transmitting irrelevant data to the persistent event channel over the network. Moreover, accomplishing this task at Event Manager also reduces computational workload at Event Channels.

Persistent Event Channel (PEC). The Persistent Event Channel handles consumer's request for subscribing to the channel, unsubscribing from the channel, publishing event messages to the channel and also delivering event from the channel. Event Channels maintain persistent data storage for event messages published by event producer. All published event requests are sent to duplicate event handlers to check for duplicate event messages to avoid network connection delay. This can be done by checking the event ID that has been assigned by event producer's application. An event with unique event ID is stored in the channel storage persistently. Each event in the channel is uniquely identified by its URL. And thus each event resource can be accessed my consumer by sending http requests using the standard http verbs such as HEAD (meta-data), GET (read), PUT (replace), POST (create and write).

B. *The mobile client*

In this architectural framework, mobile clients are thin clients such as smartphone and tablets. Applications for these devices are responsible to register themselves to a particular channel group or group of channels based on the channel topic by consuming the Pub/Sub web services hosted in the code. Once a device registers itself, it continues to receive event notifications for any updates made in the persistent channel. In order to provide code flexibility and interoperability, the client side application is designed following the Model-View-Presenter (MVP). A stub component of the backend server is hosted in the Model. The stub is responsible for all incoming and outgoing transactions.

Once an event update arrives at the stub, the latter passes the event to the View's logic through the Presenter to be displayed on interface layout. Likewise, event messages produced by client actions (e.g. button click) are passed to the stub through the Presenter which then transmits the data to the backend server.

The *Model* component of the client application is designed to contain a persistent storage for event notifications. Moreover, it contains a queue for unpublished events; events that are produced by the client actions but could not be delivered due to the connection loss. These unpublished events are removed from the queue once they are delivered to the backend server. All interactions between the *Presenter* and the *Model* take place though the stub. The major functionalities of a stub are as follows (see Figure 3):

- *Connection service.* The stub is responsible to connect mobile application to the proxy server. Whether the communication should take place over WebSocket connection or should it be http polling are decides by the stub.

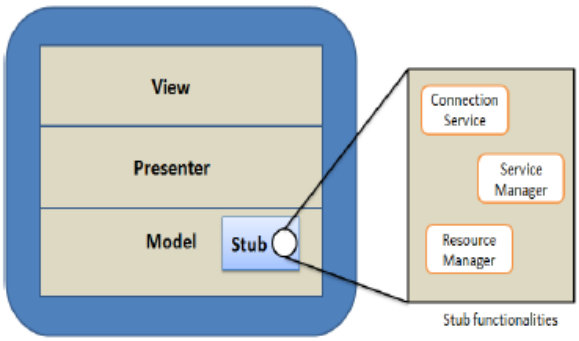

Fig. 3. Mobile client architecture

Service Manager. The stub provides the same interface of the remote cloud hosted Pub/Sub web services. It binds client's application to the remote web services over Web. It also enables client applications to invoke the consecutive functionalities of the remote web services such as subscribing to Channel, publishing data, retrieving data or unsubscribing from channel in a way as if calling to local functions. All event messages generated by these actions are encoded into JSON format before they are transmitted between client and proxy.

- *Resource Manager.* The stub is responsible to store update notifications to the local storage when it arrives from proxy. States of the stored event notifications are used to check for event updates at the proxy when client application reconnects after an intermittent connection loss. Stub also checks for the unpublished events in the queue once after every connection establishment

IV. Performance Evaluation

The experiments analysis and evaluation serve to demonstrate the framework's feasibility in various event dissemination patterns and also to identify the best performing scenario. The three major components in this experiment setup include mobile users (event producer and consumer), Pub/Sub Proxy layers (Protocol Layer, Device Layer and Event Manager), Pub/Sub Persistent Event Channels.

- Mobile client: Mobile clients are running on ASUS Transformer Prime tablet. The device specifications are Android™ 4.0 Ice Cream Sandwich OS, NVDIA® Tegra® 3 Quad-core CPU, 1 GB memory and 1.3 GHz CPU Speed.
- Pub/Sub proxies: A Windows 7 desktop machine is used to host Pub/Sub proxy layers, with 64-bit Windows 7 Professional Intel® Core ™ i5-2400 CPU, 16.0 GB Memory and3.10 GHz CPU Speed.
- Pub/Sub Persistent Event Channels: A Windows 8 desktop machine is used to host Pub/Sub event channels. The hardware specification is 64-bit Windows 8 Enterprise Intel® Core ™ i5 CPU4.0 GB 3.2 Memory, GHz CPU Speed.

A. Client app performance test

The purpose of this experiment is to observe the system's performance in request/response on different client application platforms. In this experiment, three different application platforms that have been tested are Erlang client, JavaScript Desktop browser and device embedded browser. Each of these platforms establishes WebSocket connection to its backend system.

Scenario. In this experiment, 5 kb of event messages has been published from the initial sender to the Persistent Channel and 1 kb of event messages has been pushed to mobile clients by Event Router. As the event message propagates from sender to the receiver, the Round-Trip-Time (RTT) has been observed.

Discussion. Among the three client applications, the best performance is observed on the Chrome browser running on Desktop One possible reason that the app on Android WebView performs slower than Chrome browser is because WebView is linked to the Android application layer written in Java. For every activity in WebView for example JIT (just-in-time) compilation of JavaScript, the callback function is invoked. Moreover, the integration of an external framework in the application such as Phone Gap might have added an additional execution time which in turn causes performance deterioration

B. System (protocol) overhead

This test is conducted to observe the amount of overhead the chosen dissemination approaches introduces on the system in terms of latency in consuming a resource from the Persistent Channel. The chosen approaches include client pull over HTTP Ajax and server push over WebSocket. The purpose of this test is to observe the time difference and identify which approach performs better in event dissemination.

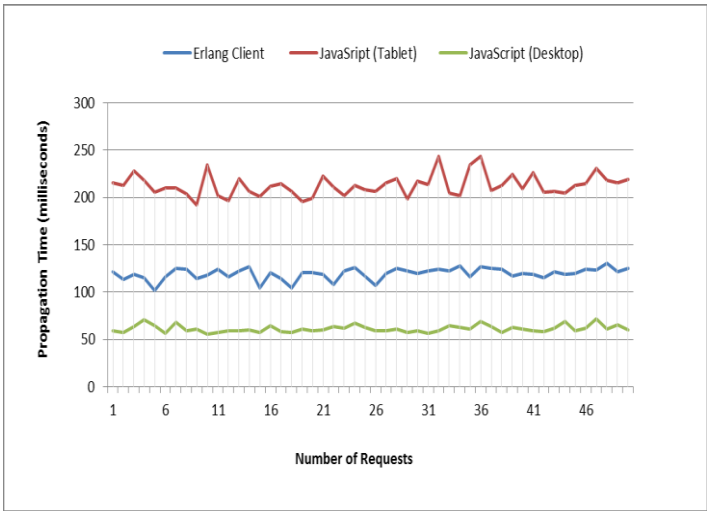

Fig. 4. RTT per request (multiple client platforms)

Scenario. The event update message is stored in the persistent channel. The experiment is conducted in two scenarios. In the first scenario, mobile consumers who are subscribing to a channel are configured to pull for event updates from the channel every 2 seconds. In the second scenario, as event updates arrives at Persistent Channel, Event Router pushes the update to the subscriber's end i.e. update propagation does not require any requests arriving from the subscribers.

Discussion. The result of client pull and server push is shown in the Figure 5. The graph shows the time for individual update propagation (50 samples) obtained from an average of five iterations where the size of each event message is 10 kb. From the graph, it can be observed that, time consumption in first scenario where the message propagates from event publisher to the server and having server send update to the subscriber as a response for update request takes much longer time comparing to the time of propagating event from publisher to the server and having server push the update to the subscriber. Time in event consumption is observed almost 1.5 times faster in server push scenario compared to client pull.

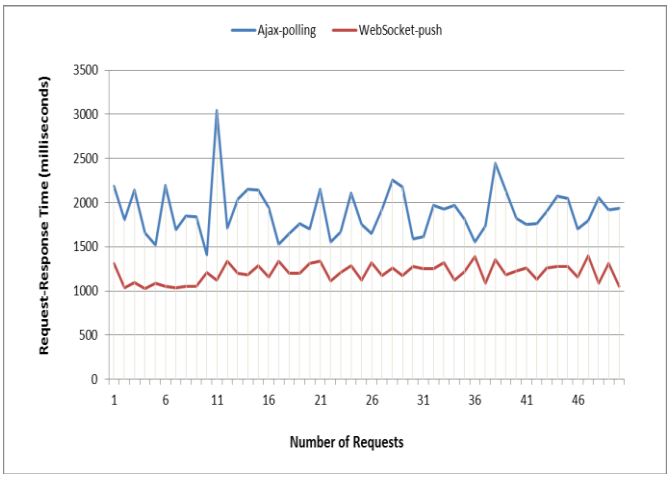

Fig. 5. Response time per request over http polling and WebSocket

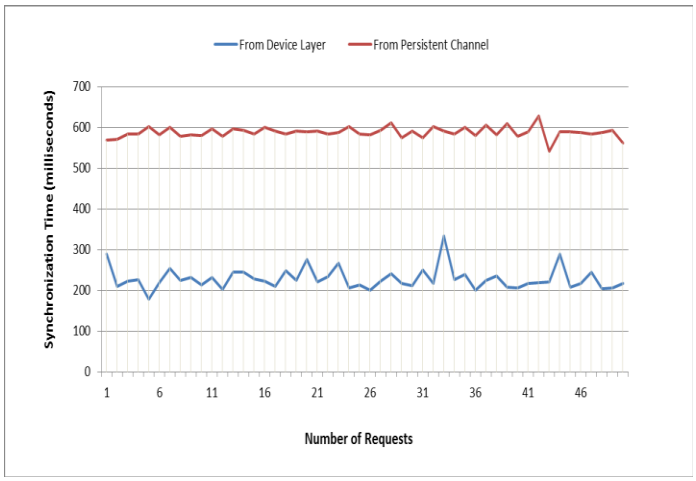

Fig. 6. Response time per request from the device layer and from Persistent Channel

C. Resource synchronization test

A framework that is designed to run part over heterogeneous network for example in this case, part over wireless network and part over LAN, one problem that arises in accessing resources from a far node is the routing overhead. In the proposed framework of this research, a client process is maintained for each individual subscriber at the device layer where the resources are stored temporarily.

If the client process is not maintained at the device layer then the alternative approach in synchronizing client side resource would be sending request for updates at the Persistent Channel which is multiple hops away from the clients. Therefore consumer's resource state can be synchronized from two different locations – Connector process of the device layer and the Persistent Event Channels. Hence, the purpose of this experiment is to observe system's performance difference in maintaining and not maintaining a client process at the device layer.

Scenario. In conducting the experiment, a resource has been published at the Persistent Channel. In first scenario, a client process with a temporary storage is maintained, hence the published resource has been pushed to the Connector by Event Router and client resource is synchronized with the backend resource at the device. In the second scenario, published resource is made available to only Persistent Channel. Hence client application is configures to synchronize its local resource at the Persistent Channel

Discussion. The results from the experiment are graphically presented in Figure 6. The graph shows the synchronization time for 50 individual requests. Each synchronization time plotted on the graph is an average time of five iterations. A resource of size 5kb has been synchronized between client's local storage and the backend storage based on client's current resource id. Results shows that the average time required to synchronize the resource from device layer is 228.5 milliseconds while it is 588 milliseconds if synchronized from the Persistent Channel Layer which is 2.6 times (157.3 %) slower. Hence, maintaining a client process in a closer proximity of the client device can result in a better performance in synchronizing data in a distributed framework.

D. Bandwith Consumption Test

This experiment analyzes the bandwidth consumption over wireless network in disseminating resource updates to the corresponding clients. The purpose of this experiment is to compare the throughput of update dissemination over traditional client pull approach with the server push based data dissemination in Pub/Sub paradigm. The experiment investigates the technique that helps in efficiently consuming available bandwidth by avoiding unnecessary network traffic in communication network. As the updates are propagated from Pub/Sub server to clients, bandwidth is calculated at server's end for every incoming and outgoing interaction.

Scenario. In this experiment, a similar scenario of System Overhead test has been adopted. This experiment is conducted in two phases. In first phase, client app is configured to send resource update request at a constant rate (i.e. every 2 seconds). Upon receiving the client request, Pub/Sub server responds with an update notification of 2kb of message payload and the updated resource. In case there is no update available, sever acknowledge the requester with a message "No update is available". In second phase, Pub/Sub server pushes the updated resource to the subscriber without subscriber prompting for the update.

Discussion. Figure 7 shows the throughput in kilobyte/second for individual resource propagation in client pull and server push approach of event dissemination. In this experiment, 10kb of data has been transferred between mobile client and server. The average throughput obtained over http polling is 5.8 kb/s when the average throughput over WebSocket is 8.6 kb/s. Bandwidth consumption over WebSocket results in at least 1.5 times higher compared to http polling.

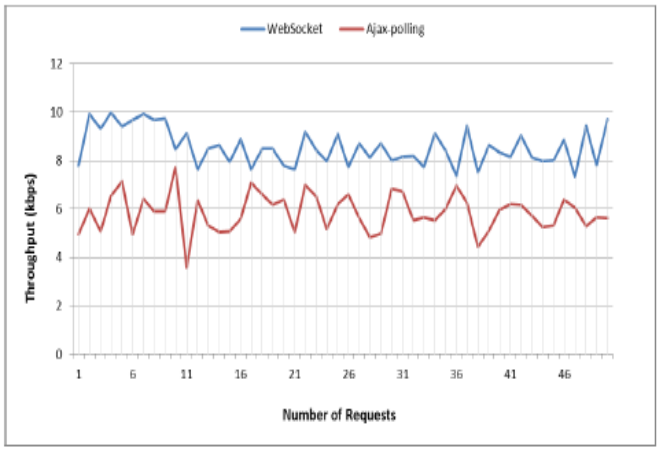

Fig. 7. Throughput per request over http polling and WebSocket

V. Conclusions and Further Developments

The research contributes in the domain of Web Services based event dissemination in Pub/Sub domain as follows: analyses different patterns of RESTful Web services within Pub/Sub domain for disseminating consumer data, studies the latest Web communication technologies and different data dissemination patterns to address the challenges of network latency in mobile environment, proposes a solution for traditional pull-based architecture by adopting WebSocket as a communication protocol and provides a platform for Pub/Sub communication on mobile environments. Further research will like to explore the following features as possible approaches that could be added to the existing framework to achieve greater performance improvements: using a decentralized Pub/Sub system, maintaining a User Profile**,** using mobile Web Service provisioning. These further developments are succinctly described in the following:

1) Decentralized Pub/Sub system. The current Pub/Sub framework is based on centralized event brokering system that relies on a single event broker. If the event broker is down then the event dissemination within the framework will be compromised hence relying on a single event broker increases the vulnerability of the entire system because it limits the system by the capacity of a single server. Hence adopting decentralized Pub/Sub model is a promising line of work. In decentralized approach, the system consists of M number of event brokers each responsible for a portion of N total subscription and hence responsible to deliver event updates to its own active subscription user's list.

*2) Maintaining a User Profile**.** The proposed framework is based on topic-based subscription scheme where users subscribe to events of a channel based on the channel topic or subject. However, subscription mechanism can be improved by introducing a subscription scheme based on the actual content of an event which provides more granularities in event subscription through offering a fine filtering mechanism on events. In this mechanism, maintaining a user profile can be useful in defining filtering rules in event subscription. Nevertheless, the proposed framework uses a flexible queuing policy where the notifications are buffered until the subscriber reconnects. A more complex and granular queuing policy would buffer undelivered notifications based on the subscriber defined propertied such as priorities and expiry dates of event channels.*

*3) Mobile Web Service Provisioning**.** One of the major trends of distributed system network and also a future direction of this research is the emergence of mobile terminals as Web Service providers also known as Mobile Hosts. When lot of research focuses on provisioning Web Services from resource constraint mobile device, some research works sees the potential of using smart and more powerful mobile devices with sufficient speed as the service delivery node in peer-to-peer settings.*

References

[1] Gartner, 2011. Gartner says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011.

[2] Lomotey, R.K.; Deters, R. "Reliable Consumption of Web Services in a Mobile-Cloud Ecosystem Using REST", 2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE), On page(s): 13 - 24, vol., no., pp.13,24, 25-28 March 2013

[3] Webber, J., Parastatidis, S., Robinson, I. 2010. REST in Practice, O'Reilly Media. Retrieved on March 20th, 2012.

[4] Liu, C., Liu, Y., Ma, X., Gao, J. 2010, An Application scheme of publish/subscribe system over clustering Mobile Ad Hoc Networks. P. 1-4.

[5] Baldoni, R. and Virgillito, A. 2005. Distributed event routing in publish/subscribe communication systems: a survey. Technical Report TR-1/06. The Computer Journal, vol.50(2), pp.444 -459

[6] Fiege, L., Muhl, G. 2000. Rebeca Event-Based Electronic Commerce Architecture, Available: http://www.gkec.informatik.tu-darmstadt.de/rebeca.

[7] Hohpe, G., Woolf, B. 2004. Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Boston. 2004.

[8] Anceaume, E., Datta, A.K., Gradinariu, M., Simon, G. 2002. Publish/subscribe scheme for mobile networks, in: Proceedings of the ACM Workshop on Principles of Mobile Computing 2002, pp. 74-81.

[9] Fiege, L.,Gartner , F.C.,Kasten , O., Zeidler , A. 2003. Supporting Mobility in Content-Based Publish/Subscribe Middleware, p.103-122.

[10] Cugola, G.,Jacobsen, H. 2002. Using publish/subscribe middleware for mobile systems.Mobile Computing and Communications Review 6(4): 25-33.

[11] Muhl, G., Ulbrich, A., Herrmann , K., Weis, T. 2004. Disseminating Information to Mobile Clients Using Publish-Subscribe. IEEE Internet Computing 8(3): 46-53.

[12] PhoneGap. 2012. Available: http://phonegap.com/

[13] Lubbers, P., Greco, F. 2010. HTML5 Web Sockets: A Quantum leap in Scalability for the Web. Available: http://soa.sys-con.com/node/1315473.

[14] WebSocket.org 2012. Available: http://www.websocket.org/

[15] Furukawa, Y. 2011.Web-based Control Application Using Websocket, ICALEPCS2011, p.673- 675.

[16] Cassetti, O., Luz, S. 2011. The WebSocket API as supporting technology for distributed and agent-driven data mining. Available: http://www.scss.tcd.ie/~casseto/NGDM11-websockets.pdf.

[17] Satyanarayanan, M., Bahl, P, Caceres, R., Davies, N. 2009. The Case for VM-Based Cloudlets in Mobile Computing, IEEE Pervasive Computing, vol. 8(4), pp. 14-23.

[18] Chun, B. G., Maniatis, P. 2009. Augmented Smartphone Applications Through Clone Cloud Execution, in Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS XII), May 2009.

[19] Ashik, K., Kazi, R., Deters, D., 2012, "Supporting the Personal Cloud", IEEE Asia Pacific Cloud Computing Congress 2012, Shenzhen, China, November 14-17, 2012.