UNIVERSITY OF CALIFORNIA

Los Angeles

Near Real-Time Push Middleware

A dissertation submitted in partial satisfaction of the requirements for the degree of

Doctor of Philosophy in Mechanical Engineering

by

Siddhartha Byron Mal

2012

ABSTRACT OF THE DISSERTATION

Near Real-Time Push Middleware

by

Siddhartha Byron Mal

Doctor of Philosophy in Mechanical Engineering

University of California, Los Angeles, 2012

Professor Rajit Gadh, Chair

The research conducted in the dissertation is motivated by the needs of near real-time video dissemination to mobile devices and Smart Grid for a near-real-time delivery and filtering system of multimedia content and sensor data. It must also be able to adapt content and data delivery for, and leverage the location determination capabilities of the newest generation of mobile Internet content consumption and generation devices, including smartphones and tablets. A novel near real-time push middleware is proposed, in which end users specify their filtering criteria by creating subscriptions, and pertinent content and data are pushed to them

automatically based on their device context parameters, including location. Its function is to receive, store, filter and disseminate content and data from and to mobile Internet devices.

Existing work has addressed the issues of publish/subscribe push data delivery and multimedia content formatting for a variety of target devices. However, the issues of fast subscription matching – specifically location-based, adaptive connection management, and a lightweight client-server push architecture for mobile devices have not been addressed. These problems must be solved in order to realize the aforementioned system. In order to quickly match multimedia content and data attributes to user subscriptions an indexing scheme must be implemented. Solutions exist for standard data (*string*, *integer*) indexing, however no adequate method exists for spatial data. A novel in-memory spatial indexing algorithm is developed that is shown to reduce content and subscription search times to a few milliseconds – a reduction of up to three orders of magnitude versus conventional database searches. Client-server architectures exist for mobile data push, however they have high transmission overhead, do not address connection management across varied networks, multimedia content formatting for varied target devices, or device location context. A lightweight client-server push architecture incorporating adaptive connection management is developed which enables client-server push channel connection continuity while traversing a variety of networks and network conditions, and for content to be pushed with very low latency to mobile and traditional clients.

The developed middleware is integrated into two projects: 1) MobiSportsLive, an instant replay video system for mobile clients and 2) a Smart Grid project for electric vehicle smart charging and demand response.

Motorola TuVista was an existing research project that investigated instant replay video creation and distribution to mobile clients. Contributions were made in multimedia content aggregation, reformatting for varied device profiles, digital rights management, and performance characterization. The deficiencies in that project, including a high latency content notification scheme and slow instant replay video creation method, drove the development and integration of the middleware into the MobiSportsLive project. In the MobiSportsLive application the middleware facilitated low latency instant replay video dissemination to mobile clients connected to a variety of networks. It was compared to the TuVista system and content notification latencies were lower by one order of magnitude.

Smart Grid will increase the efficiency and reliability of the electricity grid by adding additional sensing and control capability at the consumer and enterprise levels. Generated data must be collected, processed, and transmitted to appropriate entities for it to drive meaningful near real-time applications. For example, electric vehicle charging sensors generate current, voltage, state-of-charge, and status data. For the regional utility company to be able to monitor and control charging on a broad scale, an intermediate entity must aggregate sensor data from thousands of chargers and disseminate it to the appropriate users in near real-time. To this end, simulations were run in which the middleware aggregated user charging profiles, charger sensor data, alert subscriptions, and demand curtailment signals. The aggregated data was then used to disseminate charging data, vehicle status updates and demand response alerts to simulated clients. The middleware is shown to facilitate electric vehicle charging and low latency alerts and data updates.

The concepts, algorithms, and architectures developed in this research have the potential to significantly improve the state of the art in near real-time multimedia content dissemination and Smart Grid technology.

The dissertation of Siddhartha Byron Mal is approved.

Greg Carman

Mario Gerla

Daniel Yang

Rajit Gadh, Committee Chair

University of California, Los Angeles

2012

Table of Contents

Table of Figures

Acknowledgements

I thank my Mom and Dad for their invaluable help from grade 1 through PhD year 4. I thank my wife for providing a respite from the madness. I thank my friend Dr. Arunabh Chattopadhyay for his research collaboration and guidance. I thank Dr. Frank Shih for his good advice. I thank my friends Rudy, Abby, and Meowli for their companionship throughout the years.

I thank my advisor, Professor Gadh, for his support. I thank Professors Carman, Gerla, and Yang for serving on my committee.

Vita

Education:

1998 – 2003 **University of California Berkeley**

- BS – Electrical Engineering and Computer Science

2007 – Present **University of California Los Angeles**

- MS – Mechanical Engineering
- PhD candidate – Major Field: Manufacturing and Design, Minor Fields: Biomechanics, Structural and Solid Mechanics

Work Experience:

University of California Los Angeles (**Graduate Student Researcher) –** January 2009 – June 2009, January 2011 – Present

- Near Real-Time Push Middleware
  - Research and development.
- Motorola TuVista
  - Server-side application development.
    - Video aggregation, transcoding, digital rights management, performance and usage statistics
    - Pilot study
      - System test during live event, performance characterization
- MobiSportsLive
  - Development of mobile application, automated video clip creator
  - Integration of "Near Real-Time Push Middleware"
  - Performance characterization and comparison with TuVista
- DOE/LADWP Smart Grid Project
  - Adaptation of "Near Real-Time Push Middleware" for sensor data
  - Mobile application development
  - Electric vehicle charge scheduling algorithm design, implementation, and simulation

University of California Los Angeles (**Teaching Assistant) –** January 2008 – December 2010

- MAE184, MAE94, MAE166C

NextGen Aeronautics (Multi-Disciplined Engineer) – April 2007 – August 2007

- Embedded wireless systems development

Raytheon (Multi-Disciplined Engineer) – August 2003 – April 2007

- Real-time air-to-surface tracker
  - Algorithm design and implementation.

- Kalman filtering
- Road network model
- Coordinate transformations

Publications:

Mal, S., Gadh, R., Real-time push middleware and mobile application for electric vehicle smart charging and aggregation. *International Journal of Communication Networks and Distributed Systems, Special Issue on: Context-Aware System and Intelligent Middleware for Smart Grid*, Accepted for publication June 15, 2011.

Mal, S., Chattopadhyay, A., Yang, A., Gadh, R., Electric vehicle smart charging and vehicle-to-grid operation, *International Journal of Parallel, Emergent and Distributed Systems*, *27(3),* pp. 1-17, May 2012.

Presentations:

Smart Energy International Conference (Speaker) – October 24, 2011 – October 26, 2011

- Plugging the modern utility into a smart infrastructure
- Electric vehicle smart charging

UCLA WINMEC - Mobile Media Applications Development Hands-on Training Program (Speaker) **–** May 14, 2009

- Introduction to mobile application development, platform considerations, getting started with mobile media application development
- Mobile application interface design
- Structure of a multimedia mobile application
- Mobile video, content storage, leveraging location

Chapter 1 Introduction

Mobile internet-connected devices, including smartphones and tablets, are ideal platforms for untethered multimedia content consumption and generation. Ideally, the user would create a subscription containing his or her preferences and new content would be pushed to him or her as it becomes available. “Push” entails server-initiated delivery of content as opposed to “pull” where the user explicitly requests content. Delivery of content or data based on location is now possible since mobile devices are equipped with GPS and cell tower triangulation capabilities [85]. Specific applications in near real-time video dissemination to mobile devices, Smart Grid [83], and other enterprise applications [7,8,9,10,11,13], require near real-time filtering and push delivery of data and multimedia content based on client subscription criteria including location, to mobile and other devices. A near real-time collaborative push middleware is proposed to address these requirements, in which end users specify their filtering criteria by creating subscriptions, and pertinent content and data are pushed to them automatically based on their device context parameters, including location. Its function is to receive, store, and filter multimedia content and data, then disseminate it to mobile Internet devices. Mobile devices are constrained in their capabilities, including limited network bandwidth, intermittent network connection, screen size and resolution, human input capability, and battery life [5, 6]. These limitations present significant challenges in realizing the proposed system. Solutions exist for push email [84], however, they are not extensible to multimedia content due to its larger data size and additional attributes, e.g. resolution, bit-rate, and spatial location. Existing work has addressed image formatting for target devices with a variety of display resolutions [35, 36, 39],

however, adapting multimedia content including video in near real-time for varied device profiles and connection types and qualities has not been addressed. Architectures for push-based publish/subscribe collaborative systems for mobile devices have been researched and developed [1, 4, 5, 6, 11], but the case of fast (less than one millisecond) subscription matching for content or data with spatial location information has not been addressed.

A near real-time push middleware (NRTPM) is proposed to fulfill the requirements of near real-time video dissemination to mobile devices, and Smart Grid, to address deficiencies in existing research, and to advance the state of the art in near real-time push architectures. Its major contributions are 1) a low-overhead TCP socket based asynchronous client-server push architecture 2) an in-memory spatial indexing algorithm and 3) a resource optimized and connection adaptive mobile client architecture. The TCP socket based client-server architecture provides asynchronous, non-blocking foundation upon which the rest of the architecture relies on to push new content notifications in near real-time to a potentially large number of clients. The in-memory spatial indexing algorithm will serve to facilitate low latency content notifications by reducing search times to less than one millisecond for matching client subscriptions to content. In comparison with the traditional method of performing a database search, the spatial indexing algorithm will be able to complete a search much faster due to the “Tile” data structure used to store content, its ability to group spatially correlated content, and by virtue of residing in memory. The mobile client will make optimal use of device resources, including memory, processing capability, and battery life. It will also adapt to the type and quality of network available, as well as maintain the push socket channel while traversing different networks e.g.

3G, WiFi. Simulations are run and results are compiled comparing the push architecture, spatial indexing algorithm, and the complete middleware to existing methods and solutions.

In order to validate the NRTPM's functionality and effectiveness it is integrated and tested as part of two real-world applications: 1) MobiSportsLive, an instant replay video system for mobile clients and 2) a Smart Grid project encompassing electric vehicle (EV) smart charging and demand response.

The Motorola TuVista research project examined instant replay video creation and distribution to mobile clients. Multiple inadequacies were discovered with the project, including a high latency new content notification method, and a slow instant replay video creation process. These deficiencies motivated the development of MobiSportsLive and integration of the NRTPM. A comprehensive system for instant replay video creation, storage, and dissemination were built on the NRTPM framework. A trial was run during a live sporting event with a live video feed, automated instant replay clip creation, and human participants consuming and creating content on mobile devices connected to both cellular and WiFi data networks. Notification latencies of both systems were measured and compared.

Smart Grid will increase the efficiency and reliability of the electricity grid by adding additional sensing and control capability at the consumer and enterprise levels [53, 54, 55, 56, 57, 86]. Generated data must be collected, processed, and transmitted to appropriate entities for it to drive meaningful near real-time applications. For example, charging thousands of electric vehicles can add to peak load and require additional generating capacity [54, 55] – both undesirable effects. However, if an intermediate entity can aggregate sensor data from thousands of chargers and disseminate it to a charge scheduling entity in near real-time, the negative effects

can be mitigated and the benefits can be exploited. An EV smart charging and demand response system is developed using the NRTPM to aggregate user charging profiles, charger sensor data, alert subscriptions, and demand curtailment signals. The aggregated data is used to intelligently schedule charging by exploiting off-peak benefits, leveraging stored energy during times of high demand, and responding to demand response signals. The middleware is shown to facilitate electric vehicle smart charging, low latency alerts, and data updates.

Chapter 2 Literature Review

2.1 Push systems

Push, pull, and hybrid push/pull models for data access have been extensively researched and implemented. The Minstrel Push System [1] was one of the first research projects to implement a true push system, not simply an interval poll approach. The implementation model consisted of channels where information providers would announce the availability of new data. Each provider could have access to multiple channels corresponding to different types of data. End users can subscribe to channels of interest and all content available on that channel will be pushed to them.

The architecture is made up of three major components: 1) a Broadcaster 2) a Receiver 3) an optional Payment Server. Both the broadcaster and receiver have a Protocol Engine which handles the distribution of content over different protocols e.g. HTTP, RTP, MMS. The broadcaster manages data coming from information providers and distribution of channel information to the receiver. The information providers input rules for their content, directing it to the appropriate channel. The receiver handles client subscriptions and distributing the appropriate content from the subscribed channels from the broadcaster. The optional payment server interacts with the broadcaster and receiver to handle client payments and the subsequent distribution of the paid content.

In addition to these main components, transport system components are needed to ensure scalability and responsiveness. The transport system facilitates receiver access to content with three components: 1) a Proxy 2) a Cache 3) a Repeater (Figure 2.1). The proxy is a server that takes the receiver's request, retrieves the corresponding data from the broadcaster, and then serves it back to the receiver. It serves to protect the anonymity of the receiver and facilitate connections in the presence of firewalls. The repeater and cache both mirror the broadcaster's content and serve it to the receiver. Their advantage lies in being physically closer to the receiver thus reducing data load times. They differ in the frequency of broadcaster content update – the cache is preloaded at set times with broadcaster content, whereas the repeater is dynamically loaded with broadcaster data whenever the broadcaster receives new content from an information provider.

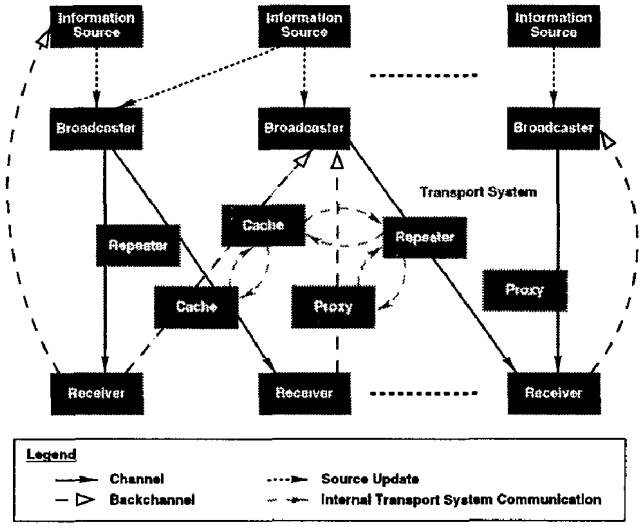

Figure 2.1 Push system components [1].

The issue of scalability is addressed in the architecture through the review of multiple broadcasting strategies and final implementation of one. Multicast using a protocol such as the

Real-time Transport Protocol (RTP) is one solution for delivering audio and video content. Multicast differs from traditional broadcast in that data packets are transmitted to a group address then replicated as needed for each receiver. Multicast is a bandwidth efficient content distribution strategy, however it is limited to protocols which support it e.g. RTP, which are currently limited to audio and video transmission. In addition it requires additional infrastructure setup, including router configuration and allocation of a multicast group address [3]. Another broadcasting solution is a regularly scheduled client initiated pull. This is not a true broadcast strategy, but accomplishes the goal of providing the client with updated data. The drawbacks include retrieved content that is not real-time if the polling frequency is not high enough, or high network traffic if the polling frequency is high enough to provide real-time data. This is one the most common methodologies used in push systems due to its simplicity, scalability, and robustness. The third broadcasting strategy is server push where the broadcaster dynamically sends data to the subscribed receivers. This requires the broadcaster to maintain a database of subscribers to contact. If the receivers must be contacted consecutively and the number of receivers is large, receivers at the beginning of the database will receive real-time data while those towards the end will receive the same data much later. This may be a problem, depending on the time sensitivity of data and number of users. The final strategy reviewed is hybrid server push and client pull. The broadcaster pushes a notification of content availability to the receiver and the receiver may then request the actual content from the broadcaster. This is the strategy used by Minstrel.

A distinction between push and event-based systems is made. The primary role of a push system is near real-time dissemination of data from broadcasters to receivers. The purpose of an

event-based system is to notify users of events, where any user may generate an event. Whereas a push system is limited in frequency and size of content dissemination, an event-based one is not due to the inherently lightweight nature of events. Event-based systems may process a high frequency of events compared to push systems because events have a small overhead while push messages have a larger overhead since they are information oriented. The concept of channels does not exist in an event system since there is no static connection between producers and consumers or broadcasters and receivers. Since any user may be a broadcaster or receiver the idea of a channel is replaced by that of a pattern. A pattern may be a single event keyword, or a sequence of events. Patterns are used to group events and consumers may receive notification upon the occurrence of a pattern.

The Minstrel push system was the first research project to present a component architecture and address the design issues of scalability and multiple push methodologies. Though the issue of scalability is discussed, a detailed solution is not proposed. Scalability is affected by many factors including maximum number of total users, maximum users supported per server, type of data being pushed, subscription complexity – none of these is addressed. In addition, push, pull, and hybrid push/pull content dissemination strategies were analyzed qualitatively but not quantitatively. The final decision to use hybrid push/pull was based solely on qualitative investigation, not on a model or experimental results.

The same authors later addressed the issue of push content dissemination for mobile users [19]. They argue that the additional problems imposed by mobile users versus fixed users are increased routing complexity and frequent disconnections. This paper uses the foundation

proposed in the Minstrel Project [1] and builds on it. The mobile push architecture proposed is show in Figure 2.2.

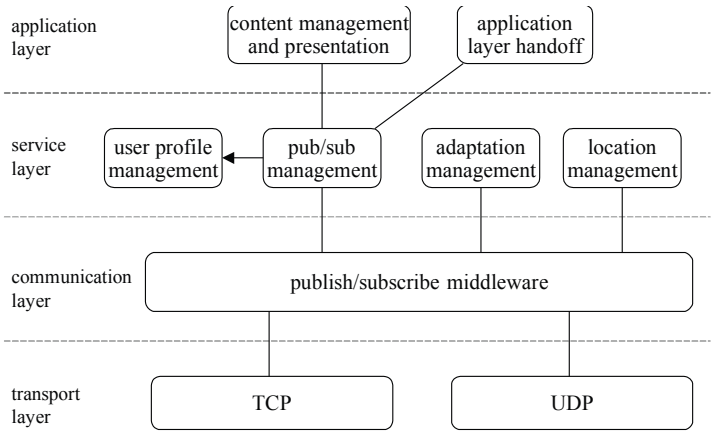

Figure 2.2 Mobile Push Architecture [19].

The application layer coordinates other services, manages the different versions of content for different devices, and transfers content between content dispatchers. The communication layer handles the interaction between publishers and subscribers, namely the publish/subscribe middleware. The service layer performs the following management functions: user profile, publish/subscribe, adaptation, and location. Publish/subscribe management handles the queuing of notifications for users who are offline, ensures notification delivery, and prevents duplicate notifications. The location management service maps the user's current device to his user profile and lets the application layer deliver the appropriate version of the content. User profile management stores rules that are used to customize content delivery for each of his mobile devices. A user can choose to not have content delivered between certain hours of the day, and also can limit the update frequency. The adaptation service processes content and reformats it for the device it is to be delivered to.

One issue in pushing content to a mobile user, ensuring delivery, is addressed thoroughly in this work. However, the issue of how the push connection is maintained between client and server is not addressed.

2.2 Publish Subscribe

Publish-subscribe is a commonly used strategy for intelligently filtering data at the network edge and allowing users to limit the information they receive to content of interest. Mühl et al [4**Error! Reference source not found.**] describe a strategy for information dissemination to mobile clients using publish-subscribe. The issues of push and pull of information are also investigated.

Information may be disseminated via push or pull periodically or non-periodically. Periodic communication entails that the provider and consumer contact each other at specified intervals. This interval may be static or adaptive. Non-periodic communication is client initiated in a pull system, or event based in a push system. This system decided to use a non-periodic push scheme for information dissemination. This decision was based on the following advantages: 1) only one message needed to communicate new data 2) provider can initiate instant communication when new data is available 3) multicast may be used instead of point to point contact 4) non-periodic pull may be used in conjunction to allow clients to update current or past content.

Specific push technologies are investigated, including JavaScript applets and pushlets. Java applets can implement push through a callback such as CORBA or RMI. These type of callbacks block on the server side which limits their scalability and makes them disadvantageous

for a large-scale push system. JavaScript pushlets establish an HTTP connection from the client to the server which is kept alive and over which data is sent from the server to the client. However, if the connection ends due to connectivity issues, the server is not longer able to push new data.

Content-based publish-subscribe is described and analyzed. The basis of publish-subscribe is that content producers asynchronously publish notifications informing consumers that new content is available, and consumers create a request for content that matches their filtering criteria called a subscription. A notification service may be used to decouple producers and consumers. Its purpose is to transmit published notifications that match consumers' subscription criteria. Filtering is accomplished using a Boolean expression of types and attributes.

For larger number of users, a distributed system is used (Figure 2.3). Each machine is a "broker" that manages the transmission of content notifications to consumers. Each broker handles an exclusive set of clients. In the simplest implementation, when a notification is issued, it is flooded to all brokers that then deliver it to clients with a matching subscription. In a more advanced implementation, each broker also has a routing table whose entries contain the subscriptions of clients and the broker that handles that client. When a notification is issued it propagates through the broker network according to routing tables. Routing tables are updated using a routing algorithm when clients update their subscriptions.

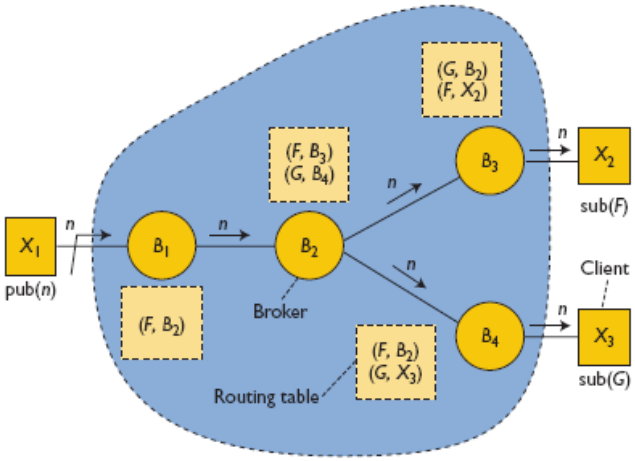

Figure 2.3 Content-base routing [4].

The specific needs of mobile clients are addressed, including automatically limiting the number of updates, allowing the client to limit update frequency, and connectivity.

Since mobile client connections are subject to frequent disconnection, a mechanism is needed to provide the client with the experience of a persistent connection -- it must prevent duplicate and missed notifications. This is achieved by duplicating the transport layer functionality of notification receipt acknowledgement. Notifications are queued until an acknowledgement is received. If the connection is dropped then re-established, the client and broker transmit the last received notification number and notifications resume. If the client reconnects to a different broker the middleware transfers the state to the new broker and the routing algorithm updates the routing tables.

The case of extended-time disconnections is handled by assigning a time limit to client subscriptions. Notifications will queue while the client is disconnected until the specified subscription time is reached, at which point the subscription will be cancelled and queued notifications will be removed. In addition clients can specify a limit to the age of notifications

they wish to receive and whether or not they only wish to only receive the most recent notification for a specific content item.

Experimental results show that if each client accepts updates at the maximum rate of the system (100 ms), the amount of traffic increases exponentially with number of users (Figure 2.4). If users limit the frequency of updates to between 100 ms and 4000 ms, network traffic increases linearly with number of users. These tests assumed one producer publishing stock quote updates every 100 ms and each client subscribed to 10 stock quotes.

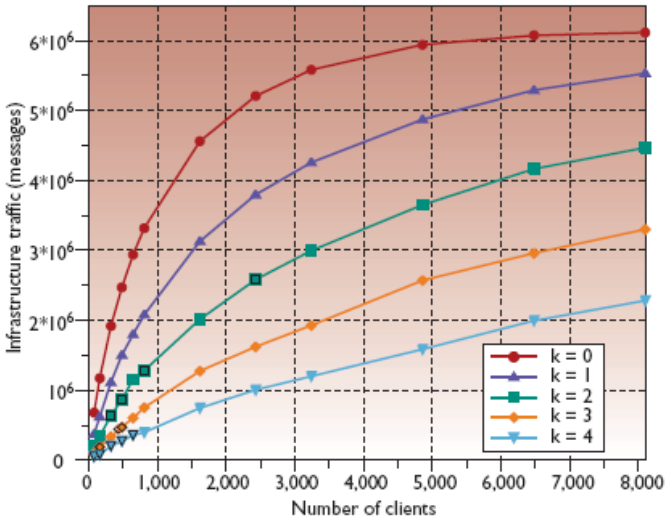

Figure 2.4 The effect of update frequency limiting [4].

This research presents a push framework for mobile clients with an emphasis on scalability. The issue of push vs. pull vs. hybrid push/pull is addressed, but only qualitatively. A distributed publish subscribe notification system for mobile devices is detailed. It takes into account the erratic nature of mobile connections by queuing notifications and preventing duplicate and missed notifications. Scalability is handled by brokers that handle mutually exclusive sets of clients. Brokers have a routing table, with subscription and client destination

pairs, that allows them to forward notifications to the correct client. This paradigm works well for simple data such as notifications, however for multimedia content it would not function well.

2.3 Collaborative Mobile Framework

The MobIME project [4] proposes a framework for mobile collaborative multimedia content generation, representation, and delivery. The unique requirements of mobile devices are considered including varied device profiles e.g. screen size, screen resolution, supported image, video, and audio formats, and varied connection capabilities.

A unified multimedia format is developed that supports images, vector graphics, text, and voice data. This unified file format (UFF) is XML based and organizes objects in the DOM structure, where the document consists of a tree of nodes and nodes are defined a priori. This UFF allows for the transmission of multiple types of content and multiple representations of the same content in a single file. The use of a UFF simplifies and improves the efficiency of system communication.

The framework is built on four layers: 1) content generation 2) communication 3) content consumption and regeneration 4) content visualization (Figure 2.5).

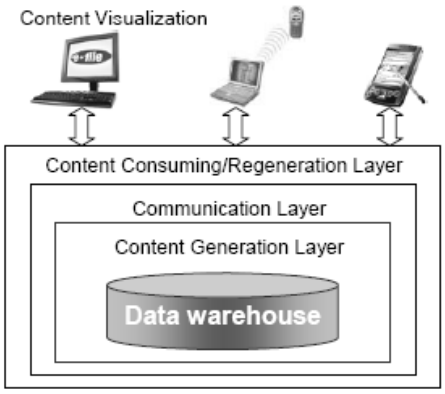

Figure 1. Framework of Mobile Collaborative Environment

Figure 2.5 Mobile collaborative framework [4].

The content generation layer generates a UFF based on the client request, which includes the requesting device profile, network status, and requested content URL. This layer is comprised of a message-oriented middleware that handles content generation and delivery. The requested content is embedded in the UFF and sent to the message agent. The message agent either delivers the content if the client is online, or stores it for later transmission when the client comes online again. This is called the smart content service.

The communication layer is comprised of a session management object and profile management object. The session management object authenticates the user when he logs in, maintains a record of client transactions, and handles connections and disconnections. The profile management object keeps track of the user device type and operating system and connection status (communication protocol, bandwidth).

The content consumption and regeneration layer is on the client side and parses the UFF received from the content generation layer and feeds it to the content visualization layer. If the

user alters the content received e.g. annotates an image with text, this layer delivers the altered content to the server for access by other users.

This work presents a layered architecture for content delivery to a variety of devices using a UFF and a message oriented middleware to handle the delivery of content. A number of mobile characteristics are accounted for including erratic connectivity and varied device profiles. The smart content service does not intelligently suggest content based on past user behavior. In addition content delivery is based on user requests, not push based.

2.4 Mobile Collaborative Middleware

Mobile Collaboration Architecture (MoCA) [6] is a middleware for developing collaborative applications for mobile users – it provides a framework for aggregating mobile users' context including location, battery life, and CPU usage. Server and client APIs are provided. The motivation for creating MoCA is the notion that mobile collaboration is not motivated by a predefined task, rather by randomly initiated sharing of information.

MoCA provides services to aggregate and disseminate device context. The monitoring daemon runs on a mobile device and sends context information, including wireless connection strength, battery life, CPU usage, free memory, current and available access points, to the context information service (CIS) running on a server. The CIS handles incoming context information from mobile devices. The CIS also receives subscriptions for notifications to be sent when device context changes. The discovery service (DS) keeps track of all applications registered with MoCA. The configuration service (CS) stores the IP addresses of the CIS and DS to be used by each mobile device – this service keeps the load equal among servers. The location

interference service (LIS) uses access point signal strength to determine an approximate device location.

MoCA provides a middleware for monitoring device context, a subscription-based notification service for alerting applications to changes in device context, and a location service for determining device position using RF signal strength from WiFi access points.

2.5 Efficient Dynamic Content Updates for Mobile Clients

Armstrong et al [14] propose a system for efficiently updating dynamic content on mobile devices. Web pages often contain dynamic content e.g. sports scores and stock quotes. Browsers employ caching and the request header "GET If-Modified-Since" to avoid unnecessarily reloading data, however non-essential content such as ads and the time of day constantly change even if the rest of the content does not. Because of this, web pages will almost always be reloaded from scratch even if the main content has not changed. To solve this problem the authors employ a client interface where the content of interest on a web page can be specified, and a resource rich proxy server that constantly polls the web page for changes in that content. By offloading the task of checking for content updates to a remote server, the client does not have to unnecessarily check the page manually for new content. This means he is free to perform other responsibilities and the battery life of the mobile device is prolonged. The means of notifying the client of updated content include polling the server, receiving push messages over a persistent socket connection initiated from the client to the server, and receiving SMS messages. When the proxy intelligently filters content updates according to a threshold change value, around 70% less data is transmitted and received in both push and poll configurations versus conventional periodic client web page reloads. The use of a proxy reduces energy usage by 50%

using WiFi and 75% using GPRS. For push and pull data access configurations the energy usage is comparable (<10% difference) for both GPRS and WiFi. Energy is measured for various data file sizes transmitted over WiFi and GPRS and it is determined that it is more energy efficient to download files larger than 30 kB using WiFi and smaller using GPRS. This is due to the large amount of power used to turn on and off the WiFi radio. This work provides quantitative evidence of the energy savings afforded by a filtering proxy as well as the difference in energy consumption between GPRS and WiFi for various file transfer sizes. However, the filtering proxy concept does not scale well to large numbers of users or large number of different web pages.

2.6 Context-Sensitive Energy-Efficient Wireless Data Transfer

Rahmati et al [16] propose an energy efficient method of data transfer by using device context including time, location, battery life, and history of WiFi access points available. The method involves intelligently choosing between WiFi and cellular data transfer based on transfer file size and estimation of WiFi network conditions without actually powered on the WiFi radio. Cellular networks require less power to stay connected, but use more energy per megabyte transferred. Energy measurements of connection establishment and data transfer for WiFi and cellular data networks are made.

Device	*Cellular (EDGE)*			*Wi-Fi*				
	Em	*Et (J/MB)*		*Ee*	*Em (J/min)*		*Et (J/MB)*	
	(J/min)	*Download*	*Upload*	*(J)*	*PSM*	*No PSM*	*Download*	*Upload*
HTC Wizard	1.2 – 6	40 – 50	95 – 125	5	19	61	5 – 7	7 – 11
HTC Tornado	1.2 – 2	100 – 150	170 – 300	10	6	53	4 – 6	5 – 7
HP iPAQ hw6925	1 – 2	130 – 160	220 - 330	13	4	46	5 – 14	6 – 15

Figure 2.6 Energy measurements [16]

A basic equation for wireless data transfer is presented (2.1), where *Ee* is the connection establishment energy, *Et* is the energy per megabyte transferred, and *S* is the transfer success rate.

$$E = Ee + \frac{n}{S} \cdot Et \tag{2.1}$$

In order to minimize the energy used to transfer *n* megabytes, the energies for transferring the data over the currently connected network and alternate networks must be calculated while taking into account the probability of finding alternate networks. A relation is developed that yields the expected energy savings for connecting to alternate network *a* (2.2),

$$E_{a,p} = P_a \cdot \left(E_p - E_{a,availabe}\right) - \left(1 - P_a\right) \cdot E_{a,unavailable} \tag{2.2}$$

where P_a is the probability that network *a* is available, C_p and C_a are the signal strengths of the respective network, E_p is given by

$$E_p = \frac{n}{S_p(\vec{C}_p)} \cdot Et_p(\vec{C}_p) \tag{2.3}$$

$E_{a,available}$ is given by

$$E_{a,available} = \frac{n}{S_a(\vec{C}_a)} \cdot Et_a(\vec{C}_a) + Ee_a \tag{2.4}$$

and $E_{a,unavailable}$ is given by

$$E_{a,unavailable} = Ee_a \tag{2.5}$$

The expected energy savings for each data transfer is calculated and if the value is positive the device connects to the alternate network and the transfer is performed. In real world tests from 14 participants the algorithm provided an average of 39% energy savings for wireless data transfers.

2.7 Indexing

Zhou et al present a Hybrid Index Structure for Location-Based Web Search [20]. It is noted that location-based searches are two-dimensional in Euclidean space and conventional text searches are set oriented. A hybrid index structure that considers both the geographical and textual information on a page is proposed. The geographical scope of a page, textual place names, is extracted then represented as a two-dimensional spatial object. The spatial object used to represent a location is a minimum bounding rectangle (MBR). The R*-tree is used to index the MBRs (Figure 2.7).

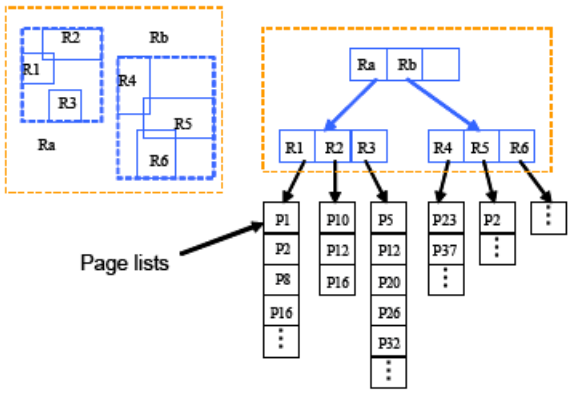

Figure 2.7 R*-tree structure [20]

Each leaf node of the R*-tree is a region represented by an MBR that points to a list of pages whose geographical scope is included by this MBR.

Chapter 3 Near Real-Time Push Middleware

In order to address the issues present in the reviewed literature, fulfill the requirements of and advance the state of the art in multimedia push systems for mobile devices, the near real-time push middleware (NRTPM) is proposed

3.1 Architecture Overview

A five layer architecture is proposed based on the work of Su et al [5] with the addition of layers to facilitate mobile push and content management (Figure 3.1). Layers are separated according to major function performed in the overall architecture. Each layer is logically separate from the others and performs a specific function.

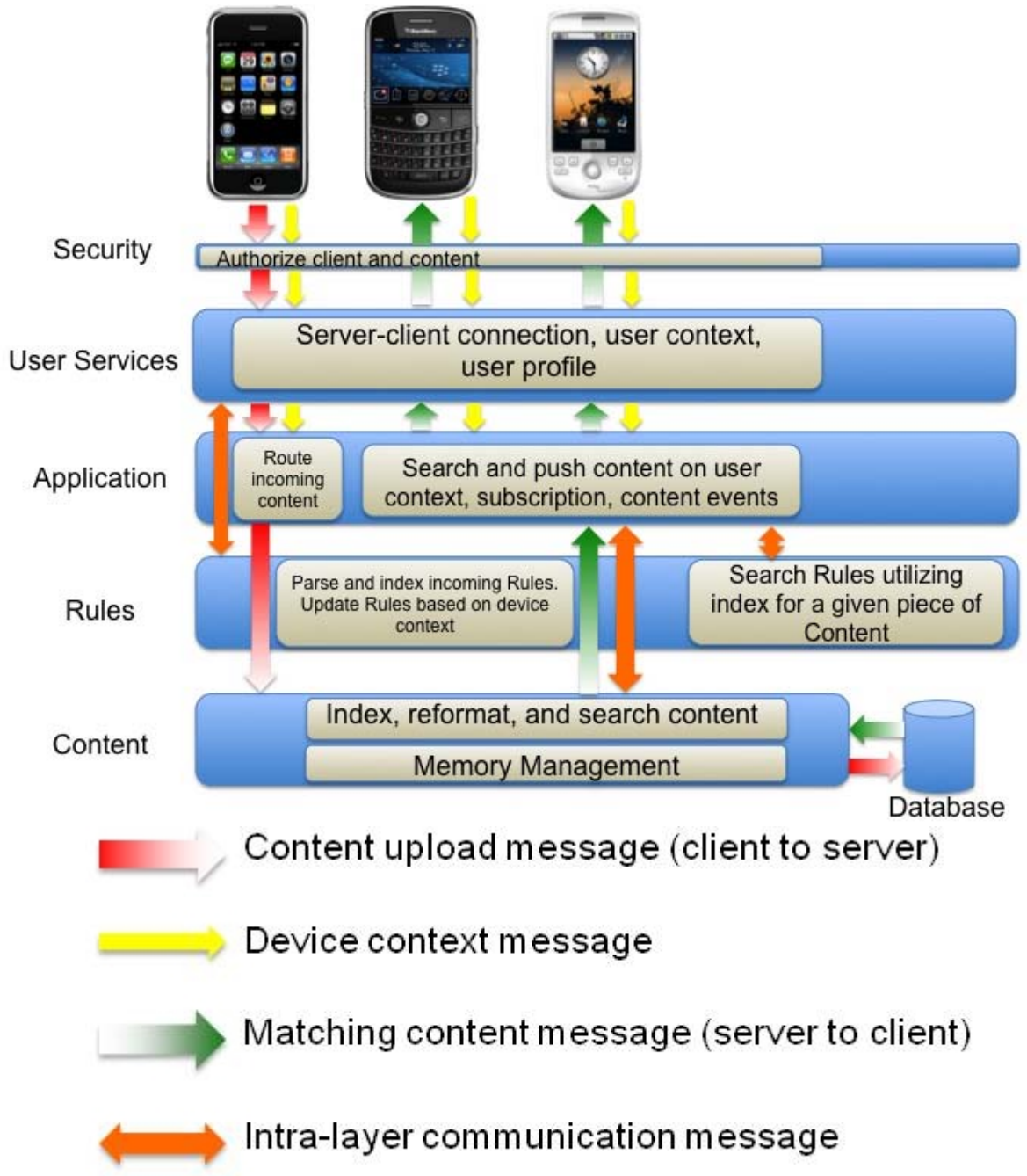

Figure 3.1 NRTPM 5 Layer Architecture.

3.2 Security Layer

The security layer authorizes new and existing users, authorizes client content requests and uploads, protects content from unauthorized users, and disallows access to banned users. All

mobile devices have a unique device identifier (UDID) that can be accessed using an API call. The client application transmits this number in an encrypted format to the server the first time it is run. An API key is generated for this particular user then sent back to the requesting client application. The client is authorized using the hash of this key, the request URI, and a shared secret [17]. If a certain user has been banned, or an enterprise requires that only a known set of devices can access the system, a list of allowed or disallowed UDIDs is maintained by the security layer that enables it to restrict access. After the first-run authorization, the application will initiate subsequent sessions using the same authorization scheme. To avoid the need to authorize every request by computing a hash, which becomes computationally intensive for a large number of requests, the security layer will issue a session token with a defined time of validity [21]. Every request issued by the client will include this token and as long as it matches a token in the list of currently valid tokens, the request will be authorized and fulfilled. The session token will be renewed while the user is still connected, and will expire if the user logs out, quits the application, or if the expiration time is exceed and the user is not connected.

3.3 User Services Layer

To realize a middleware that can push near real-time data to devices including mobile phones and tablets, the connectivity issues associated with these devices must be addressed. Mobile devices can connect to a variety of wireless data networks including cellular 2G, 3G, 4G LTE, WiFi, and WiMAX. The device's IP address may change at any time due to physical movement between cell towers or switching between networks caused by a change in availability of networks. The second issue is the intermittent nature of wireless data connections. A mobile device's connection is subject to random disconnection because of physical phenomena like

radio interference and hand-off issues between towers. To contend with these issues the User Services Layer (USL) must handle user connections and disconnections, keep track of user connection status, ensure content delivery, and collect and store user profile information.

3.3.1 Connection modalities

Three separate push modalities were considered: 1) TCP socket 2) UDP socket 3) Comet servlet. The first employed a TCP socket connection from a native mobile client application to a server -- the socket remains open until the client explicitly disconnects or is disconnected as a result of a network disturbance. The server application was implemented using Java NIO libraries to build an I/O multiplexed server that can handle thousands of simultaneous connections while avoiding the pitfalls of a threaded approach [24]. A server that spawns a new thread for each new connection does not scale well, as the number of threads grows large a majority of CPU time is spent handling thread overhead (e.g. context switching) and not performing useful work. The feature of the NIO libraries that makes I/O multiplexing possible is non-blocking I/O by use of selectors. The selector monitors the listening socket and creates keys based on the type of event (incoming connection, incoming data, outgoing data). The server application then performs the appropriate action according to the type of key created. A TCP type socket was chosen because mobile clients using cellular data services are often behind firewalls inaccessible to them -- the server cannot send data to the client unless an existing socket has been initiated and connected from the client to the server and remains open. For this reason a connectionless UDP socket would not work.

The second push modality considered was UDP socket. UDP is a connectionless protocol that provides no reliability guarantee [22]. Since it is connectionless, server initiated push is not

straightforward. Since a mobile client may be behind one or more firewalls and or network address translators (NATs), the server cannot simply send him a datagram as it will simply be discarded by a firewall or be lost by an intermediate NAT due to incorrect IP or port addressing. This problem can be circumvented by implementing "UDP punch through" [23]. This entails that both parties know each other's public and private endpoints (IP and port combination). The client sends a datagram to the server's public and private endpoints thus punching a hole in its firewall. The firewall will now assume a datagram coming from the server's endpoint and addressed to the client's endpoint is a response to the client's initial datagram. Implementation of UDP punch through presents several drawbacks: 1) additional communication for client and server to inform each other of public and private endpoints 2) additional data transfer to initiate punch through 3) process must be repeated when the client changes network.

The third push modality was the Comet model which enables asynchronous data delivery from server to client by employing a long-held HTTP request. The client initiates an HTTP request to the server which is held until data becomes available. When the server has data to send to the client it is sent in the response message, called an event. After receiving the response from the server the client must send a new HTTP request, assuming he is interested in receiving further events.

The advantages of the socket server over the Comet web server are less memory and processor intensive server, smaller message sizes, ability to handle thousands of simultaneous users, and shorter content arrival latency. Its drawbacks include a greater level of effort for integration with a web application, and the necessity for a developer implement to implement more functionality that would already exist in a web server. The advantages of the Comet Servlet

are that many HTTP servers already exist, including Apache Tomcat, that are mature and support the Comet model and that the client may connect using a web application as opposed to a native device application. Its disadvantages include the inability to handle thousands of users due to its one thread per client model, and an increased delay in content receipt at the client end. We define the delay to send an HTTP request to be *dr*, which is the time taken for the request message to travel from the client to the server, and the content notification delay *dc*, the time to transmit a typical content notification from the server to the client. We assume $dr \geq 10dc$ since both messages are less than 1000 bytes, but the uplink bandwidth of the client is usually at least an order of magnitude smaller than that of the server e.g. for a 3G mobile client 1 Mbps vs. 10 Mbps on the server-side. The worst cast delay for the Comet client will be $dr+dc$, where a new content notification is queued to be sent right after the server finishes sending one, and the best case will be *dc*, where the user has been connected for some time and a new content notification has just been enqueued. In both cases we assume an underlying persistent connection and that the socket client has already connected at some time in the past. In addition to a longer delay in new content notification using the Comet method, more energy is used by the device to continuously send HTTP requests after receiving content notifications.

TCP socket was chosen for the connection modality between clients and the server since it provides the best balance of resource utilization and speed among the three.

3.3.2 Content synchronization

In order to maintain content synchronization between the client and server, the intermittent nature of mobile connections and the behavior of TCP sockets must be considered. Though the transport layer (TCP) already provides an acknowledgement scheme, this functionality is not

accessible at the application layer. Moreover, if the client has disconnected ungracefully e.g. process crash, intermediate router reboot, the server is left with a half-open connection. TCP assumes that once a socket is connected it stays connected and active unless some event shows otherwise. In this case a write operation on the socket may succeed but an IOException may be thrown later when message delivery times out or fails, at which point we have no knowledge of the message that was sent. To guarantee content delivery to mobile clients on intermittent connections, it is imperative that a message queueing and acknowledgement system be implemented in order to maintain client server content coherence.

When a message is sent to a client it is added to a HashMap called *messagesPendingACKQueue* (Figure 3.2). This HashMap uses the clientID as the key and a first-in first out queue of QueuedMessages, which has been implemented as a circular list. (Figure 3.2) Once the client receives the message he adds the message number to a list of acknowledgements to send with the periodic alive message. Since it requires more energy to send an acknowledgement (ACK) message to the server for each content notification received than to send them all at once, and in the interest of client battery energy preservation, the message acknowledgements are bundled with the periodic alive message. Once the server receives the bundled acknowledgements, it looks up that client's ID in the *messagesPendingACKQueue* and removes the QueuedMessages with the corresponding numbers. If the client disconnects before sending its combined alive/ACK message then subsequently reconnects, it immediately sends the combined message. Messages that remain in the queue without receiving an acknowledgement for more than the maximum alive message

period are resent up to a maximum of five times. Messages are discarded if the client has disconnected and does not reconnect within five minutes.

Figure 3.2"Messages pending acknowledgement" data structure.

3.3.3 Adaptive Connection Manager

The client is responsible for initiating and maintaining its connection to the server – the Adaptive Connection Manger (ACM) handles this task. When the socket connection between a mobile device and a remote server is interrupted, it can be broadly attributed to one of two causes: 1) the local network connection between the mobile device and a WiFi access point or the WWAN has been disrupted 2) the internet connection between the local access point and the remote server has been disrupted (Figure 3.3). The former is most likely to be disrupted – causes include interference, cell site signal saturation, handoff failure, absence of signal, and transitions between different networks. The Mobile IP standard [25, 26, 27] provides a protocol whereby mobile devices can traverse varied networks and maintain the same IP address, however, this has not yet been widely implemented. In addition, when a user is connected via WWAN his IP may be reassigned at any time by PPP [27]. The ACM addresses these deficiencies in the current network protocols.

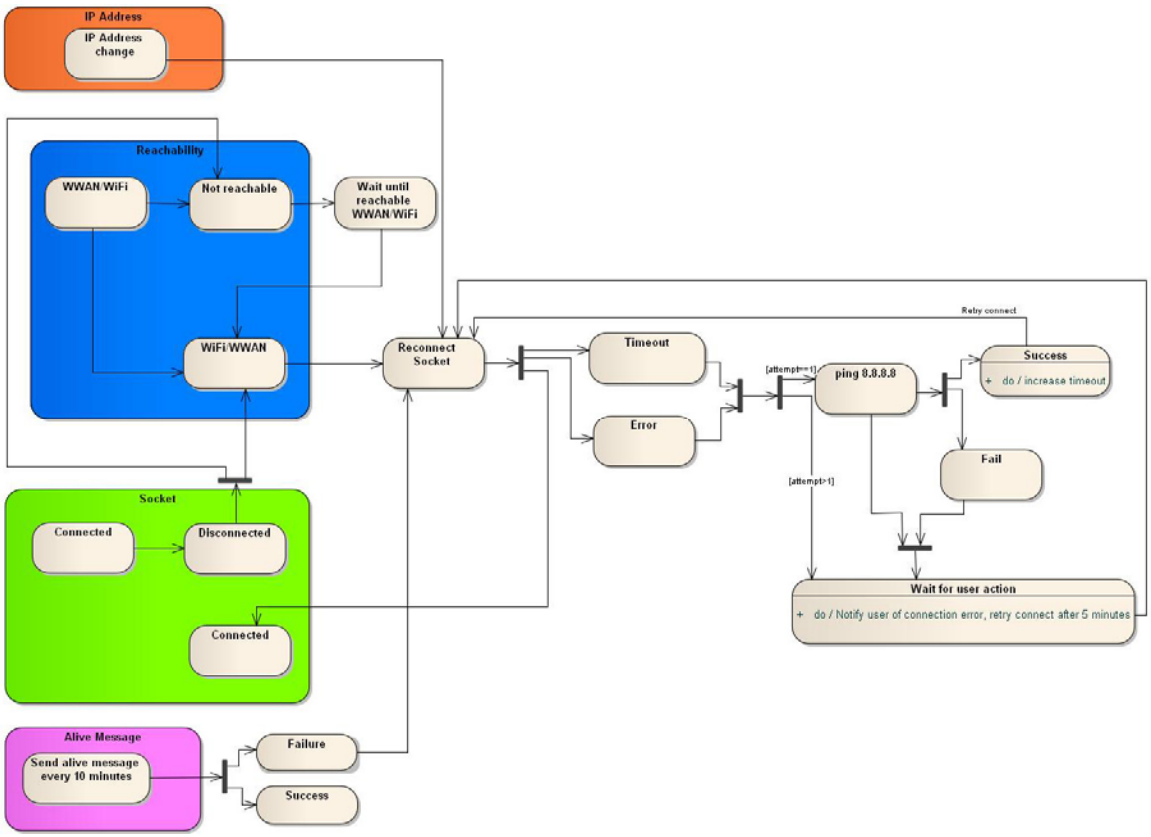

Figure 3.3 Adaptive Connection Manager logical functionality.

3.3.3.1 Connection disruptions

3.3.3.1.1 Local network connection disruption

To detect a local network connection disruption the mobile device's Reachability API is used. This API provides functionality to determine if data destined for the server can leave the device i.e. if a packet can make the first hop on the route from the mobile device to the remote server. Once the class has been instantiated, the current "reachability" state is determined i.e. can the first hop be made and over what type of network. The class notifies a specified observer of any subsequent changes to that state including when the connecting network type changes e.g. from WiFi to WWAN or when the first hop can no longer be made. If the connected network changes

from WiFi to WWAN or vice versa, the socket must be reconnected. If neither type of network is reachable the ACM must wait until it becomes reachable again and only then can attempt to reconnect the socket.

3.3.3.1.2 Broader Internet connection disruption

The second type of disruption occurs between the local network access point and the remote host. This can be attributed to an intermediate router reset or failure, or physical link failure. To best detect this occurrence, an "alive" message is sent to the server. If the server responds with an ACK message the connection is fine. If it times out or throws an exception, and the local network connection is reachable, a host that is known to be up is pinged – the Google public DNS server 8.8.8.8 is used. If the ping fails to receive a reply then there is a problem with the local network connection e.g. the device is connected to a WiFi access point but must authenticate before accessing the internet. The user is notified of a local connection error and the app waits until the user addresses the issue and manually reconnects. If the user does nothing the ACM attempts to reconnect after five minutes – this handles the case where the wireless link to the local network was poor but has improved, e.g. weak WiFi/WWAN signal. If the ping receives a reply the broader internet is reachable and the ACM attempts to connect to the server again with a longer timeout. If the subsequent connection fails the host is considered unreachable and the ACM notifies the user. The ACM attempts to reconnect in five minutes – this handles the cases where the server's connection has temporarily gone down, the server application has crashed, or a link on the route to the server has temporarily failed.

3.3.3.1.3 Socket disruption

The socket itself can end up in a half-open state even if neither the local network nor the broader internet connections are disrupted. Once a TCP socket connection has been established from the client to the server, both sides assume it is open and ready to send/receive data. The socket must be explicitly closed by either the client or server in which case a close message is sent to the other end and the socket is closed on both ends -- this is called a graceful disconnection. If an ungraceful disconnection occurs because of a process crash, intermediate router reboot, or change of IP address, the connection becomes out of sync and is called half-open. In this case an application will only know the connection has been broken if it attempts a write operation.

The server application will be informed of a half-open socket when it attempts to send a new content notification message to the client. The client will only know its socket is in a half-open state is it attempts a write operation. It is in the client's interest to detect if its connection to the server is still active. To this end the client sends a periodic alive message to the server. If the socket is broken, the write operation will immediately throw an exception and the client will attempt to reconnect. If the server detects that a client has disconnected ungracefully, it keeps his UserSubscription active and queues all content notifications until he reconnects.

The socket connection can also be disrupted if the device is assigned a new IP address, which may occur if the user is connected to a WWAN [27]. This condition is periodically checked for by comparison of a previous and current IP address. It may also be caught by the periodic alive message.

3.3.4 User profile

Each user has a UserProfile comprised of his userID, a DeviceProfile, and a UserSubscription. Everytime he connects the UserSubscription is created -- his DeviceProfile and DeviceContext are constructed from the data supplied at the time of connection, while his UserSubscription is deserialized from disk (assuming it exists). UserProfile data is available as read-only to other layers through public getter methods.

3.3.4.1 Device profile

Since a client may connect from a variety of devices using the same account, the system must know a minimum of information about the connecting device in order to provide the appropriate version of content. Once a user has connected and is authorized, the client application sends a device profile message including operating system name and version, device model, and display resolution. Missing information is looked up in a device information database and filled in.

3.3.4.2 User subscription

The UserSubscription consists of all of a user's Rules. When a client creates a rule on his device it is synchronized with the server, parsed into the Rule data-type, and added to his UserSubscription. This data is persistent among connection sessions -- it is serialized to disk when he disconnects and deserialized from disk upon connection.

3.4 Application Layer

The Application Layer (AL) functions as a coordinator and delegator -- it instantiates all the other layers and passes messages among them. Messages are processed in the USL and passed off to their corresponding handlers in the AL. The four major events that the AL coordinates: 1) when new content arrives the AL calls the RL's search function, which returns a list of clients

whose subscriptions match the new content. The AL sends a new content notification message to all clients that had a Rule that matched the new content. 2) When a Rule is changed or a new Rule is added, the AL passes it to the RL for parsing and then to the CL to search for matching content which is then sent to the corresponding client in the form of new content notification(s). 3) When clients connect, disconnect, and reconnect the AL informs the RL to activate or deactivate the UserSubscription for that user, and handles searching of content. 4) When a user sends an updated location the AL informs the RL which alters the corresponding Rules that have a "My Location" attribute. The AL makes use of threading to achieve concurrent and asynchronous processing of events and data (Figure 3.4).

When a new content item is received, the AL calls the RL's searchRules(Content c) function to determine if there are any Rules that match it. Since this may be a time consuming task, it is started in its own thread and it calls the AL's rulesSearchComplete(List<Integer> clientIDList) function with a list of clientIDs when it completes. The RL search function returns matching clientIDs to the AL that dispatches a new content notification to those clients. Concurrent with this, the AL calls the CL's addContentToDBAndIndex(Content c) in its own thread (Figure 3.4

When a Rule is changed or added or activated, the AL starts a thread for the RL to parse and index it. When the thread completes the AL starts a new thread calling the CL's searchContent(Rule r) function to find content matching the Rule. When the CL search thread completes, the AL's contentSearchComplete(List<Content> contentList) function is called with a list of matching Content, which it notifies the client of. When clients connect or reconnect after a brief disconnection, the AL passes the deserialized UserSubscription from the USL to the RL

which is then activated.

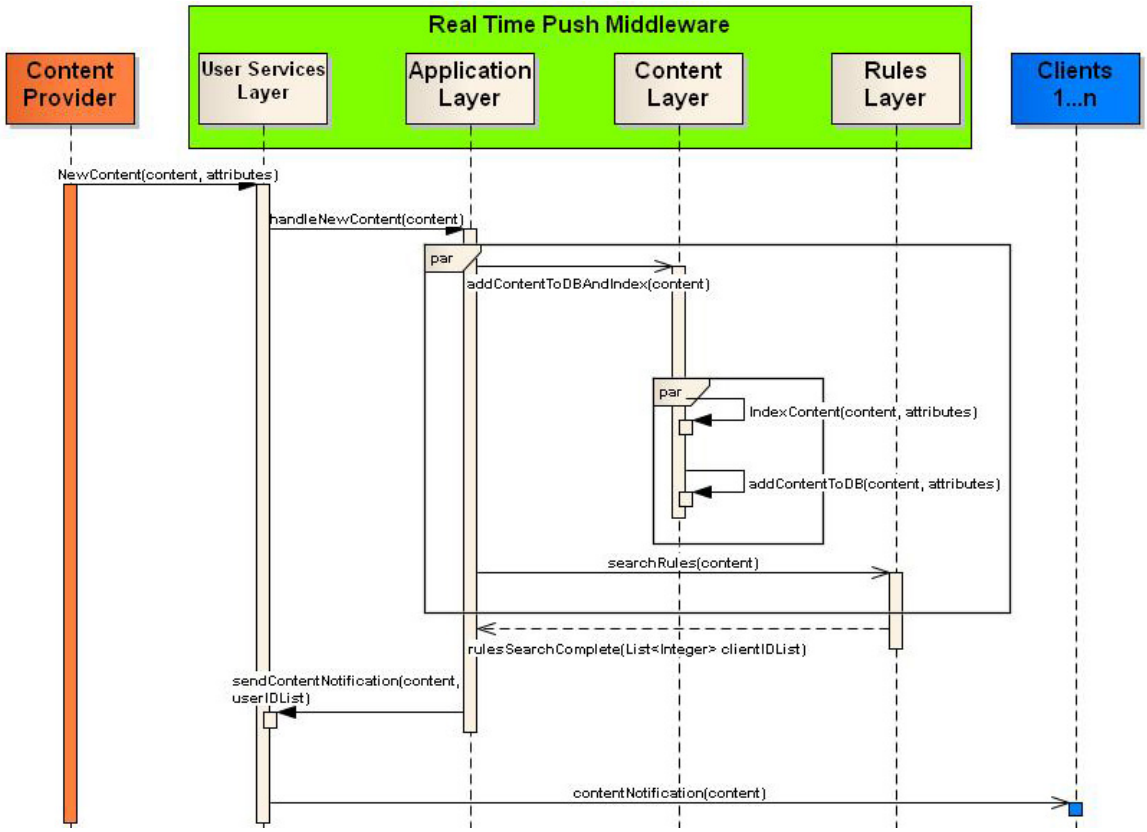

Figure 3.4 Application Layer sequence diagram.

3.5 Rules Layer

A rules engine allows for customizable solutions using logical rules specific to the problem space [49,50]. XML has become the standard for rule languages thanks to the simplicity it brings in organizing the regular expressions, conditions, values, and logic that comprise rules [50, 51, 52]. The primary use for rules in this middleware is data filtration. End users specify the types, conditions, and values that constitute pertinent data to them, and the middleware filters all incoming data according to these rules.

The primary functions of the Rules Layer (RL) are to parse new and updated rules from XML format to the Rule data structure, index the Rule in the location domain, search for matching Rules given a new piece of data, and handle serialization to and deserialization from disk of Rules.

3.5.1 Rule data structure

Rules are constructed of one to many DataTypeConditions which are in turn made up of a single data-type specifier, a field name specifier that corresponds to the database field, a condition, a value, and any associated logic (Figure 3.5). Enumerations are defined for accepted data-types, logic, and conditions. In addition a Rule has a unique name, the userID of its owner, and a unique ruleID.

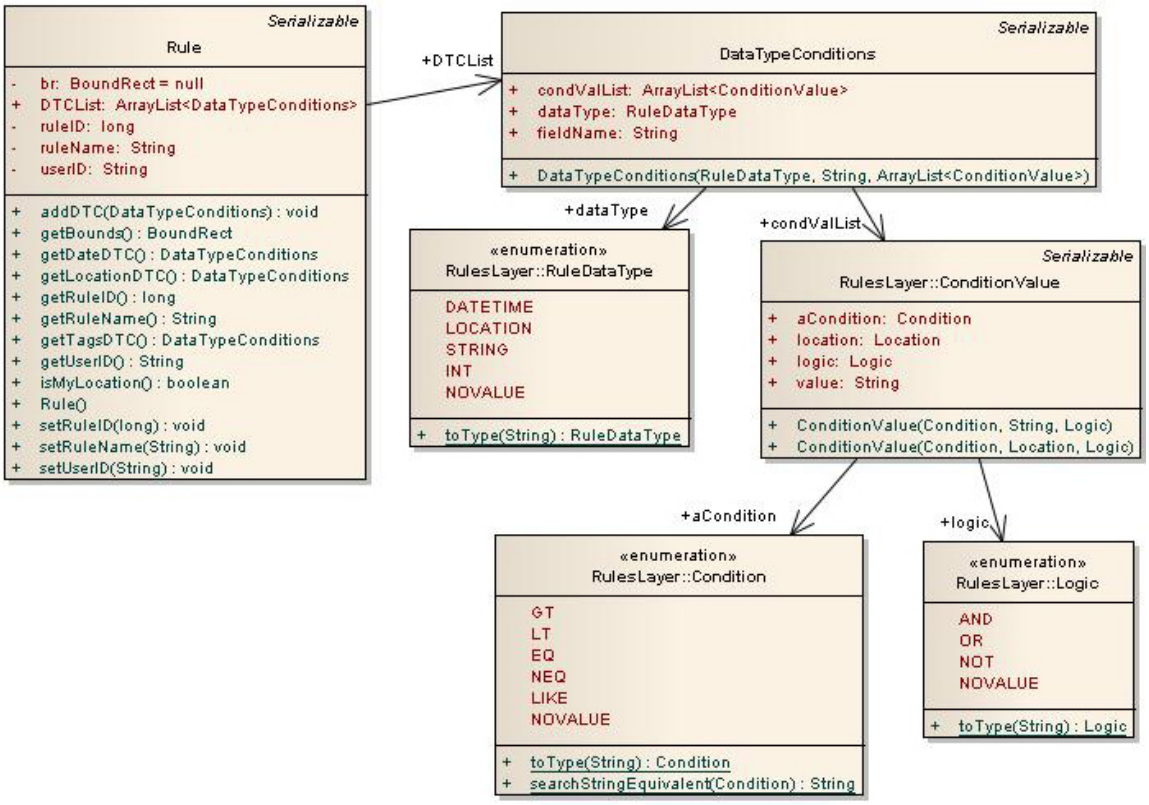

Figure 3.5 Rule class and supporting classes.

3.5.1.1 Data types

The data types that comprise DataTypeConditions are *integer*, *string*, *datetime*, and *Location*. Strings and integers are necessary to describe content characteristics e.g. meta-data tags. Datetime was chosen because of its cross-platform capability between SQL and Java, and to represent the date and time the content was created. The Location data type represents a rectangle specified by an upper left and lower right latitude longitude pair.

3.5.1.2 Conditions and logic

For each data type there are associated conditions which may be imposed. For integers conditions include greater than, less than, equal, and not equal to. For strings the only condition

that can be applied is “like”. Greater than, less than, and equal to may be applied to the datetime data type. Location can be specified as equal to or not equal to. When multiple conditions are imposed, the Boolean logic operators AND, OR, and NOT may be used to combine them. Rules are validated to ensure that ConditionValues and their associated logic will return at least one match.

3.5.2 Rule parsing and processing

Rules are sent to the middleware in XML format, validated, parsed, indexed, added to the user’s profile, and added to the database. The Java XML DOM library is used to parse the XML data into a document object model (DOM) representation. Values for each data-type are checked for validity and logic is checked to disallow Rules that can never return a match. As the XML representation is parsed, one to many ConditionValue objects are created which make up a single DataTypeCondtion object, which in turn constitute a Rule object. The user’s ID is stored in the Rule as well as a unique, monotonically increasing ruleID. Once the Rule object is created, it is indexed using its location condition. Finally the Rule is added to the list of Rules that make up a user’s subscription and is added to the database.

3.5.3 Indexing

For every new piece of content received by the middleware, all Rules for all users must be searched for a match based on the attributes of the content and the conditions, values, and logic of the Rules. Indexing of Rules is needed to mitigate search times as the number of Rules and the frequency of received content increase. The “LOCATION” data-type is chosen for indexing since spatial geography searches in an SQL database are the slowest among the *string*, *int*, *datetime*, and *geography* data-types.

The earth's surface is tessellated into Tiles of size one degree latitude by one degree longitude which are stored in a two dimensional array. The "LOCATION" condition for a Rule is specified by a bounding box which is defined by an upper left and lower right latitude, longitude pair. Rules are added to every Tile that they cover partially or fully. Tiles store Rules in a modified R-Tree structure where the bounding box of both the leaf nodes and non-leaf nodes belongs to a Rule. Figure 3.6 and Figure 3.7 show an example wherein six distinct Rules with six different location requirements are indexed using the R-Tree. Bounding boxes R1 and R2 are root nodes, R3 and R4 are children of R1, R5 is a child of R2, and R6 is a child of R4. The R-Tree structure belongs to the Tile that spans latitudes 0.0 to 1.0 and longitudes 0.0 to 1.0. Each time the middleware receives a new piece of content the RL will be queried for any matching Rules given the content location. It will in turn look up the Tile corresponding to the content location and query it for matching Rules. The Tile will search the root nodes of the R-Tree, descend any node that contains the content, and return all Rules that contain it. The SQL database will be queried for the Rules that match the IDs returned by the Tile-location search and that match the rest of the *string*, *int*, and *datetime* attributes of the received content.

This method for Rule matching is faster than a SQL location search because Tile access is $O(1)$ since they are stored in a constant sized array. If for a given piece of content there are no Rules stored in the corresponding Tile, the search time is nearly zero. If Rules are stored in the Tile, only the root nodes of the R-Tree must be searched before concluding there are no matches -- potentially significantly fewer items to search than the total number of Rules. On average R-Tree search time is $O(\log n)$ Once the Tile returns matching Rules, which are identified by their ID value, we significantly improve the performance of our subsequent SQL search because we

obviate the need for a slow SQL location search. If the ID column is the primary key for the table i.e. there is a clustered index for the ID column, a search over a Rule's ID, *string*, *int*, and *datetime* attributes will be faster than a search over the *location*, *string*, *int*, and *datetime* attributes.

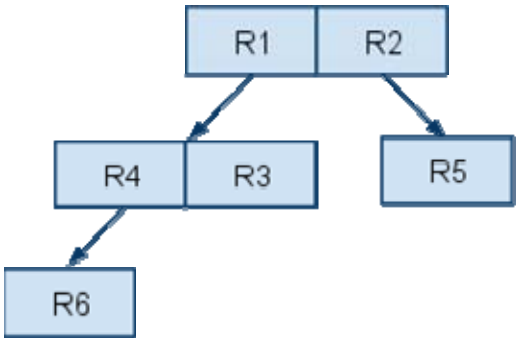

Figure 3.6 R-Tree structure.

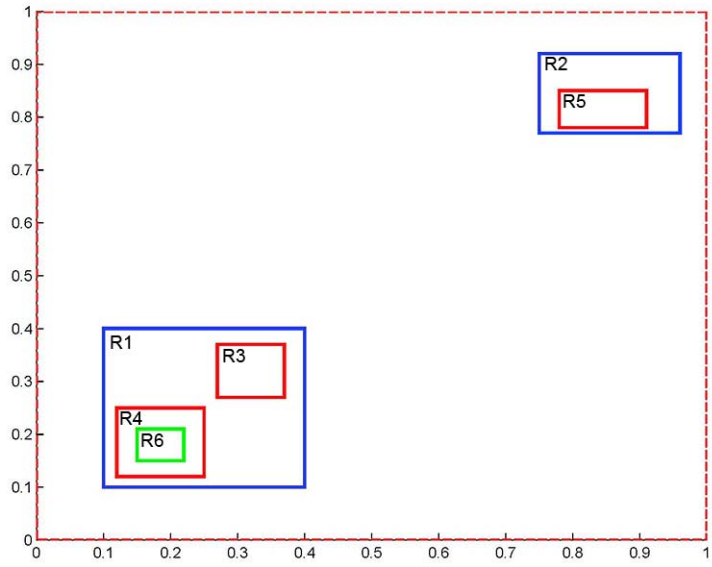

Figure 3.7 R-Tree spatial visualization.

3.5.4 Search

The RL is responsible for searching for Rules that are satisfied given a piece of content. Rule IDs are the primary key i.e. clustered index in our database yielding an average search time of $O(\log_t n)$ where t is the minimization factor. This is an improvement over the $O(n)$ search time of a non-indexed column. A String of Rule IDs is compiled from the location index and the String of IDs for Rules without location requirements. If the content has a location attribute the location index is consulted and a String of matching Rule IDs will be returned and added to our Rule ID String. Next, irrespective of whether or not the content has a location attribute, the String of Rule IDs for Rules with no location requirement is added to our ID String. A PreparedStatement is constructed using the String of Rule IDs and attributes of the content, excluding location, and the database is queried. By using the location index we succeed in narrowing down our search to a smaller subset of all the active Rules in the middleware. Search time is decreased significantly in two ways: 1) if the location index returns null we obviate a database search and the total search time is $O(1)$ i.e. constant 2) we avoid performing a SQL location search by searching over the Rule IDs, which are the content table's primary key, returned by the location index which takes an average time of $O(\log_t n)$.

3.6 Content Layer

The Content Layer (CL) handles content uploads, insertion of new content into the database, reformatting of content for target devices and varied network connections, indexing of content in the location domain, and searching for matching content given a Rule. Content may be uploaded by both dedicated content providers and clients.

3.6.1 Architecture

The CL is comprised of three distinct components: 1) Content Upload and Access Server (CUAS) 2) Content Reformatter (CR) 3) Content Layer Class (CLC). The CUAS is a Java servlet that runs on a dedicated web server and handles content uploads and content access. The CR is a standalone program that handles content formatting for supported devices. The CLC is implemented as its own class that is part of the NRTPM and handles database insertion and retrieval, location indexing, and searching. Communication among the three is over TCP/IP socket.

The CUAS runs as a servlet on a web server that resides on a machine with a high bandwidth link. Clients upload Base64 encoded [29] content, since binary data cannot be sent over HTTP. The CUAS decodes the content, creates a thumbnail image, saves the original to disk, and then hands it off to the CR for reformatting. When a client requests to access content, the CUAS authorizes the user through the security layer, determines which version of the content to serve them, and then returns the content in Base64 form in an HTTP response message. Most mobile devices consume video content in chunks using byte-ranges, so the servlet has been implemented to accommodate range requests as well as standard content requests.

Users may connect to the middleware from a variety of devices including smartphones, tablets, and laptops. These devices may connect over an array of networks including 2G, 3G and 4G cellular networks, WiFi, and ethernet. Based on the target device and its connectivity the middleware will serve a particular version of a content item.

The CR handles the reformatting of incoming content depending on its MIME type. Images are resized for a variety of popular device screen resolutions including HVGA and

WVGA. The Java Abstract Window Toolkit library is used for fast and smooth image resizing. All resizing tasks for each content item are completed in their own thread. Video is transcoded using the H264 codec to a set resolution of 320x480, which is supported by all target devices, using three different bitrates, 100, 500, and 1000 kbps, corresponding to low, medium, and high quality. FFmpeg, an open source transcoding utility is used for transcoding. Multiple ffmpeg processes are run concurrently up to a maximum, depending on the memory and CPU of the machine. The maximum number of concurrent processes is a multiple of the total number of video options being transcoded, which for this implementation is three. Since these are CPU and storage intensive processes, a servlet running on a dedicated machine handles content upload, storage, and content reformatting. Once reformatting has completed, the servlet notifies the middleware of the new content.

Once content has been uploaded to the CUAS and reformatted by the CR, its meta-data is passed to the CLC and stored using a relational database. Each content item has a unique monotonically increasing identification number which is stored as an 8 byte integer with a maximum value of 2^{63}-1.

3.6.2 Indexing

When a new Rule is received or an existing Rule is updated, all content must be searched. Additionally when a user connects, his subscription is made active and a search must be performed. Similar to the rationale behind Rule indexing, we index the content's location attribute.

The data structure used for Rule indexing, in which the earth's surface is tessellated into Tiles stored in a two-dimensional array, is also used for content. However, since a content item's

location can be represented by a single latitude longitude point, content is stored in an ArrayList within the Tile. A two-dimensional array, called contentBins, internal to the Tile data structure is used to tessellate the Tile into a 10x10 grid and count the number of content items in a particular subregion of the Tile. When content is indexed, its location is used to access a Tile in the two-dimensional array. When it is added to that Tile, a check is performed to see if any sub-Tiles contain the content. If there is a sub-Tile that contains the content, its addContent method is called with the content. If not, a check is performed to ensure this Tile's *MAXCONTENT* is not exceeded. Once both checks are passed, the content is added to the Tile's content list and the corresponding bin in *contentBins* is incremented. If *MAXCONTENT* is exceeded the createSubtile method is called.

3.6.2.1 Sub-Tile creation

The purpose of having a maximum limit on number of content items per Tile is to limit search times. Once that limit has been exceeded, the Tile should be divided into one to many smaller Tiles containing all or part of the Tile's content. Depending on the spatial distribution of content within the Tile, the sub-Tile creation process will either divide the Tile equally or create one or more Tiles containing clusters of content. In this way, spatially correlated content is grouped and can be searched quickly.

3.6.2.1.1 Determining content spatial distribution

Content spatial distribution is tested for uniformity using a χ^2 test [28]. A known uniform sampling distribution is compared.

$$
\begin{aligned}
&numBinsRows = numBinsCols \\
&E_{ij} = MAXCONTENT / (numBinsRows \cdot numBinsCols), \\
&\{1 \le i \le numBinsRows, 1 \le j \le numBinsCols\} \\
&\chi^2 = \sum_{i=1}^{numBinsRows} \sum_{j=1}^{numBinsCols} \frac{(O_{ij} - E_{ij})^2}{E_i}
\end{aligned}
\tag{3.1}
$$

Using the UnknownDistributionChiSquareTest from the Apache Common Mathematics Library[30], the significance value *p* is calculated for χ^2 using a uniform distribution and the observed *contentBins*. The *p*-value is the lowest significance level at which the null hypothesis, that the Tile's content distribution is uniform, can be rejected. The threshold significance value, *PTHRESHOLD*, chosen was 0.95 which corresponds to 95% confidence that the distribution is uniform. If the calculated value is greater than or equal to *PTHRESHOLD*, the uniform distribution hypothesis is accepted and the Tile's content is determined to be uniformly distributed. Otherwise the distribution is assumed non-uniform.

3.6.2.1.2 Uniform content distribution

If the Tile to be sub-Tiled has been determined to contain uniformly distributed content, it is divided into *n* equally sized sub-Tiles. If *n* is square, i.e. $\sqrt{n} \in \mathbb{Z}$, then the number of Tiles along the longitude is the same as the number along the latitude (3.2).

$$
nLat = nLon = \sqrt{n} \tag{3.2}
$$

If *n* is not square, i.e. $\sqrt{n} \notin \mathbb{Z}$, the number of Tiles along either the latitude or longitude dimension is equal to the largest prime factor of *n* (3.3).

$$n = p_1 \cdot p_2 \cdot \ldots \cdot p_t$$
$$nLat = p_t, nLon = n / p_t \quad \textbf{(3.3)}$$

The length and width of each sub-Tile are calculated as a fraction of the original Tile's dimensions (3.4).

$$W = currentTileWidth / nLon$$
$$L = currentTileHeight / nLat \quad \textbf{(3.4)}$$

The sub-Tiles are created, added to the current Tile's list of sub-Tiles, and content is moved to the corresponding sub-Tile (Figure 3.8).

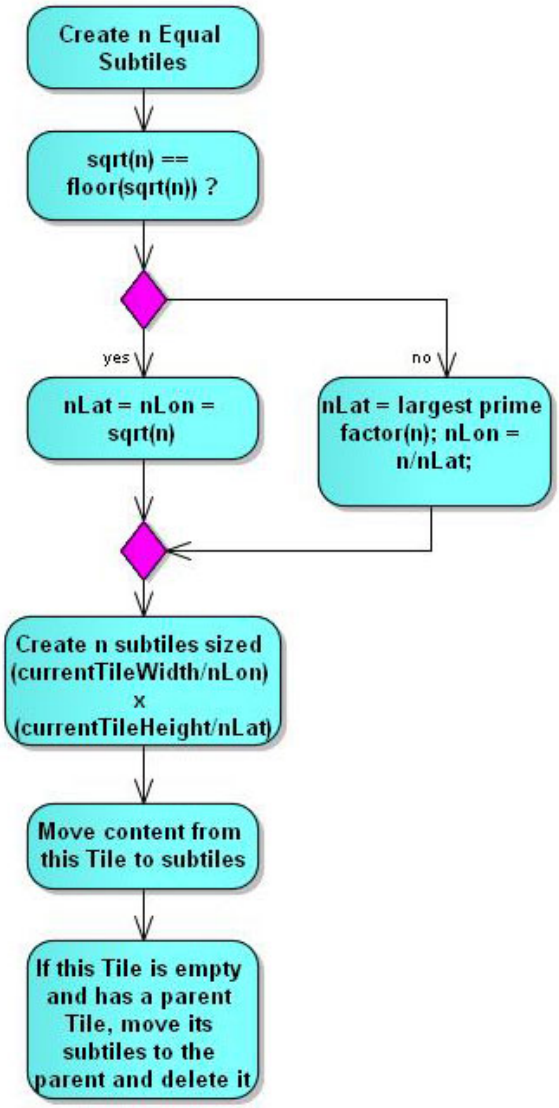

Figure 3.8 "Create n equal sub-Tiles" activity diagram.

3.6.2.1.3 Non-uniform content distribution

3.6.2.1.3.1 Cluster identification

Clusters are defined as groups of content that are highly spatially correlated. The criterion for identifying a cluster is content items within *maxClusterDist* of a chosen cluster centroid. *maxClusterDist* is defined as $1/15^{th}$ of the longest dimension of the current Tile.

$$maxClusterDist = (h_{Tile} > w_{Tile})\,?\,h_{Tile}/15 : w_{Tile}/15 \quad \textbf{(3.5)}$$

Potential cluster centroids are chosen at the center of every *contentBin* that contains more content than the median for all *contentBins*. The Tile's *contentList* is then iterated over and content is added to the closest centroid that is within *maxClusterDist*. The list of potentials is pared down by removing any clusters that have no content. Clusters whose centroids are within 2**maxClusterDist* of each other are merged and their new centroid is placed at the center of mass of the new merged cluster (Figure 3.9).

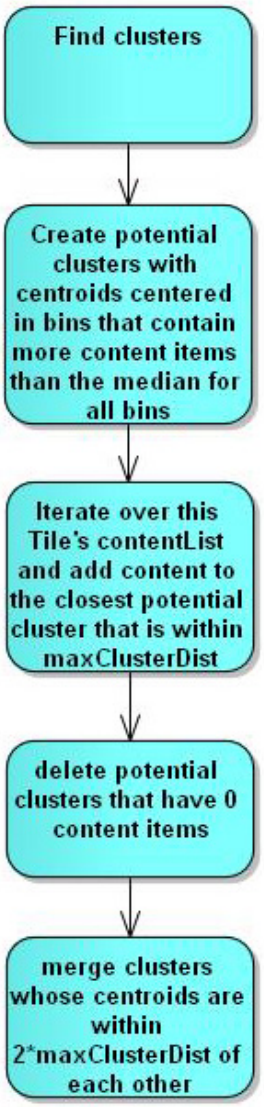

Figure 3.9 "Find clusters" activity diagram.

3.6.2.1.3.2 Sub-Tile creation containing one or more clusters

Once a list of potential clusters has been identified, the sub-Tile creation function creates potential Tiles that contain each potential cluster. Next potential sub-Tiles are created that contain multiple clusters. The list of candidates is pared down by removing Tiles that intersect

existing sub-Tiles, Tiles that do not have a minimum amount of content, *minSubtileContent*, and Tiles that are larger than a maximum area, *maxSubtileArea*. The potential Tiles are sorted accorded to their content density i.e. the ratio of number of content items to area. A new list of selected Sub-Tiles is created starting with the head of the potential list, which is the Tile with the best content to area ratio. Subsequent Tiles are selected if they do not overlap previous selections. If the sum of all selected subTiles' content is less than 50% of the current Tile's *MAXCONTENT*, the values of *minSubtileContent* and *maxSubtileArea* are respectively reduced and increased by 10% and the potential sub-Tiles are iterated over until their total contained content is at least 50% of *MAXCONTENT*. The initial value of *minSubtileContent* is 10% of MAXCONTENT and *maxSubtileArea* is 10% of the current Tile's area (Figure 3.10).

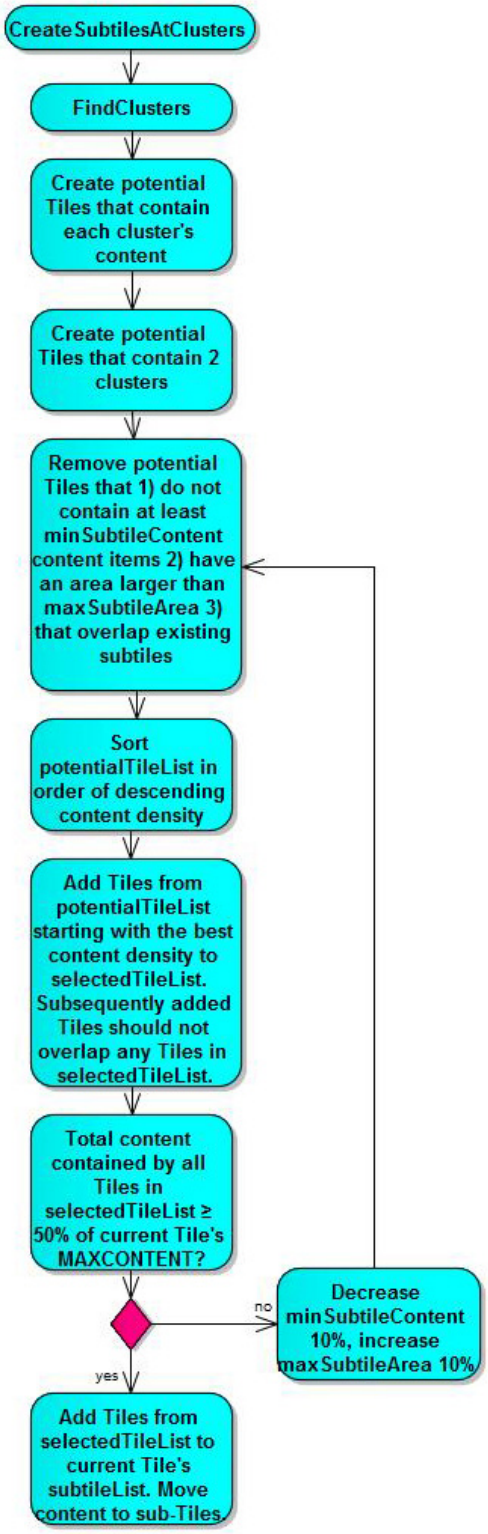

Figure 3.10 "Create sub-Tiles at clusters" activity diagram.

3.6.2.2 Memory Management

The amount of memory used by the location index is a tweakable runtime parameter that is set based on available system memory. When the memory limit is reached, the CL serializes Tiles to disk in order of least-recent last access date/time. If a Tile needs to be accessed and has been written to disk, the CL deserializes the disk version back into memory. Serialization and deserialization times depend on the number of objects, including content items and sub-Tiles, stored in a Tile.

In the interest of content freshness, references are only stored in Tiles for a set amount of time before they expire and are removed. The expiration time is a tweakable runtime parameter that is set based on content freshness requirements and content or data update frequency.

3.6.3 Search

Content must be searched when a Rule is modified, or added, and whenever a client connects/reconnects and his subscription is activated. Assuming a Rule has a location condition, the location index is queried and will either return a null match or a list of all matching *contentIDs*. If the Rule contains additional parameters, such as a *string* or *datetime* condition, the SQL database is queried using the *contentIDs* returned from the index, and the additional Rule parameters.

Index searches are much faster than database searches since Tile access is $O(1)$ i.e. constant time. If a database search is necessary, using the reduced search space of *contentIDs* returned by the index decreases search times as well. The database's *content* table has the

contentID field as its primary key. A clustered index is created for the primary key, *contentID*, which means it is structured as a B-Tree and its search time is at worst $O(\log n)$ and on average $O(\log_t n)$, where t is the minimization factor [31, 32]. Use of an in memory index decreases search times for all search scenarios. When the index returns null a database searched is made unnecessary and search time is $O(1)$. When the index returns a list of *contentID*s, the search space is greatly reduced and search time is reduced since a database location search does not have to be performed, and the *contentID* is the table's primary key.

Chapter 4 Results

4.1 Real-Time Content Delivery Performance

In order to deliver content in near real-time, it is imperative that the client and server have a lightweight messaging architecture. The processing of sending and receiving messages on the server-side must be very fast. The latency, processing time, and energy consumption of push and poll content notifications are measured and compared.

4.1.1 Push vs. Poll

4.1.1.1 Latency

New content uploads constitute a stochastic process -- content may be uploaded at any time to the server by any authorized user or dedicated content producer. To quantify the real-time content delivery performance, the latency between when the middleware receives a new content message and when the middleware is notified that the client has received the message is measured.

Tests comparing poll and push content delivery methods were performed using a content provider sending new content at a random interval between 200 milliseconds and 5 seconds. For the push method one client stayed connected via a TCP socket throughout the duration of the test. For the poll method the client polled the server every 200 milliseconds via HTTP request. The client had one Rule comprising his subscription. Five trials were conducted for each method with the same seeded random interval generator i.e. same new content update interval sequence among tests. In order to determine when the client receives the new content message, an

acknowledgement message is sent from the client to the middleware – under normal operation message acknowledgements are not sent after receiving every message, in the interest of battery life. The ping time from client to server during these tests averaged 28 milliseconds with a standard deviation of 1.8 milliseconds.

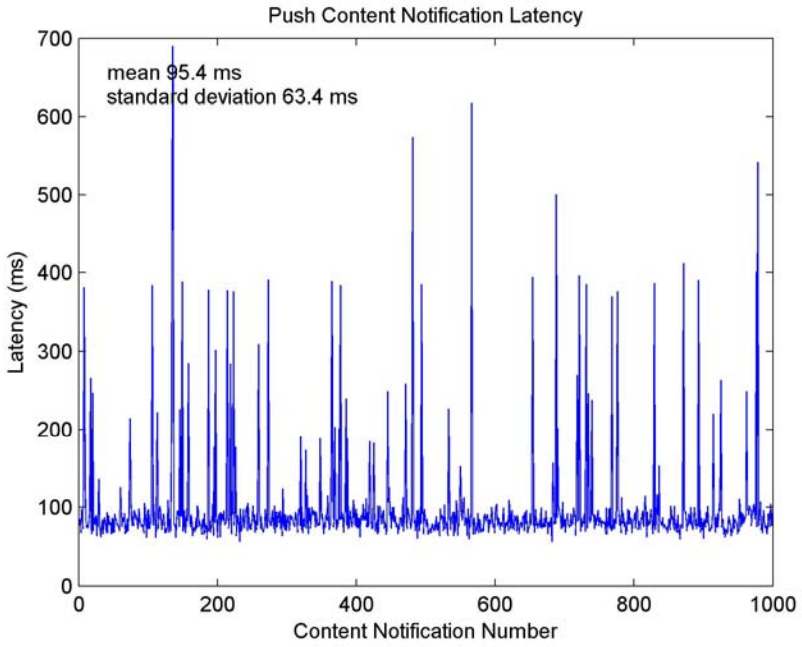

Figure 4.1 Push content notification latency.

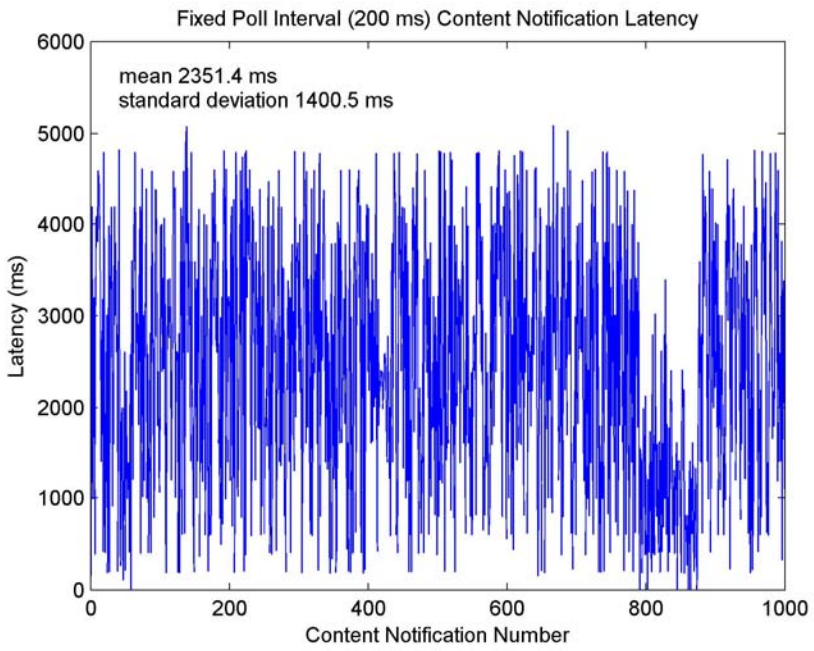

Figure 4.2 Fixed poll interval (200 ms) content notification latency.

Latency for the push method was measured from the time of arrival of the content at the server to the receipt of the acknowledgement message from the client that it received the content notification. Average latency was 95.4 milliseconds with a standard deviation of 63.4 milliseconds (Figure 4.1). The high, in relation to the mean, standard deviation value can be attributed to the intermittent spikes in latency. The spikes can be attributed to several factors including variations in network speed and processing time for new content. If there is any decrease in network speed due to congestion or other factors, the latency will increase. If multiple new content messages arrive at the middleware nearly simultaneously, the latency will be higher for those received later due to the associated processing time for each new content item. Although new content processing is threaded, the benefits of concurrency are limited by the machine's number of physical processors or cores, and other processes running, among other factors. Due to this, when multiple new content items need to be processed simultaneously, they

will complete in the chronological order they were received. Latency for the poll method was measured from the time of the first new content request message to when the response contained new content. Average latency for this method was 2351.4 milliseconds with a standard deviation of 1400.5 milliseconds (Figure 4.2). The latency for push content delivery is lower by almost two orders of magnitude than the poll method. This can be attributed to the push delivery method being able to send content as soon as it arrives at the server, as well as the lightweight messaging architecture of the client and server.

4.1.1.2 Processing Time

Only one client with a one Rule subscription and one content provider were used in order to minimize the contribution of Rule and content processing time to the test. The notification message was 26504 bytes, the acknowledgement message was 9 bytes, the downlink speed was 4.6 Mbps, and the uplink speed was 6.2 Mbps, the combined transfer time for the messages themselves was 46 milliseconds. Ping time, which is a measure of network latency, from client to server averaged 28 milliseconds. The sum of the transfer time for messages to and from the client and network latency was 74 milliseconds. The difference between the network travel time of the messages from the average content notification latency yields a processing time of 21 milliseconds. This figure includes processing the new content notification from the provider, searching the client's subscription Rule and matching the new content, and server and client message processing. A direct measurement of the processing time taken for sending the notification message and receiving the client's acknowledge message yielded a value of zero milliseconds. The same value was recorded for the client processing the new content notification and sending an acknowledgement message. This indicates that the time taken was less than 1

millisecond, which signifies that overhead from client and server message processing is negligible.

4.1.1.3 Battery Life

In addition to latency advantages, push content delivery also holds device battery life advantages over the poll method. In the traditional data delivery model the client has to continuously poll the server for new content. Since new content arrives asynchronously at the server some of the new content poll requests will return no new data. On a mobile device this has the disadvantage of wastefully draining battery energy.

Battery state of charge was measured using a fully charged (battery state of charge 100%) iPhone for push and poll methods for a total of 4500 seconds with new content being added randomly between 200 milliseconds to 5 seconds and a fixed poll interval of 200 milliseconds (Figure 4.3, Figure 4.4). Since the iPhone battery level API call only returns the state of charge in 5% increments, the final state of charge was interpolated assuming a linear depletion rate.

At the end of the testing period, the push method depleted the battery's state of charge by 55% less than the poll method. Since polling has no foreknowledge of content arrival at the server, much energy is expended sending requests that return no new – in contrast push only sends when there is new content thereby wasting no battery energy.

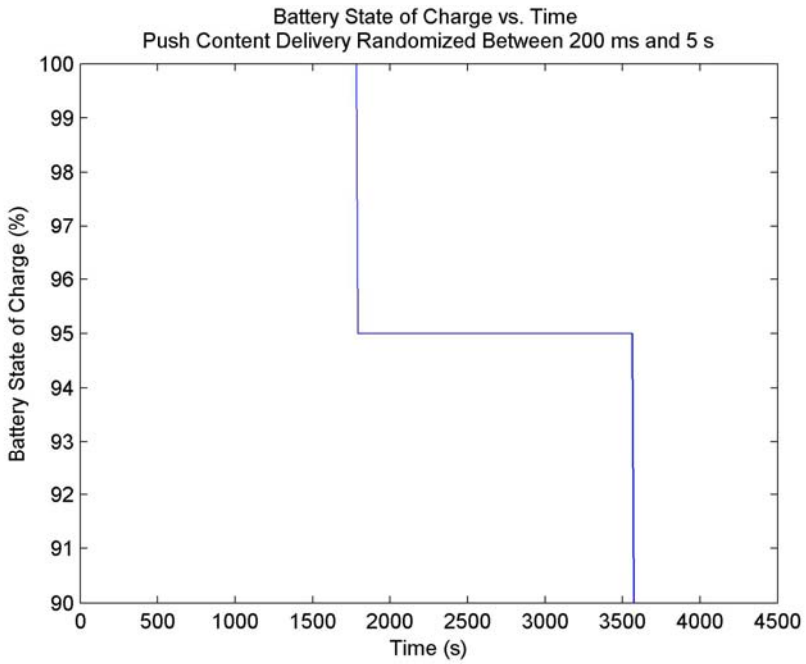

Figure 4.3 Battery state of charge versus time - push content delivery, content updates randomized between 200 ms and 5 s.

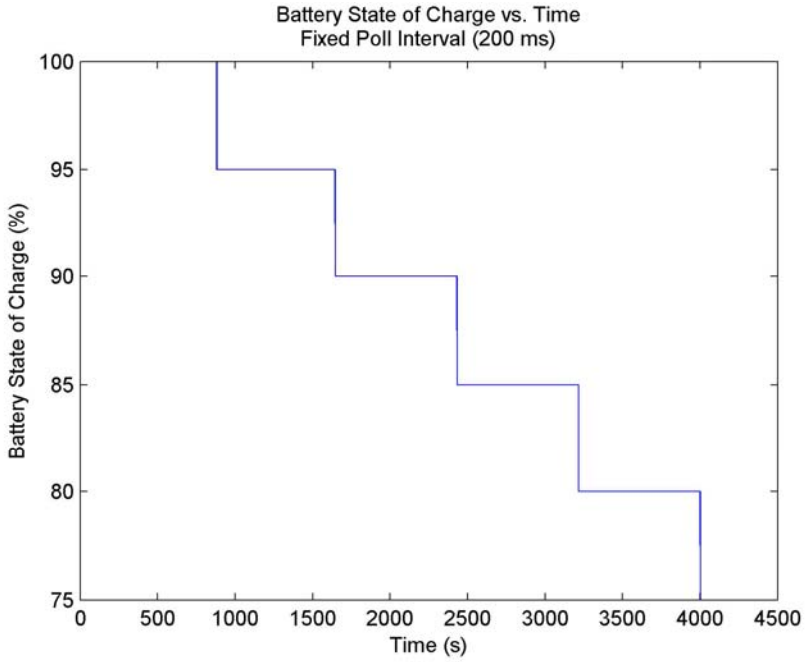

Figure 4.4 Battery state of charge versus time – 200 ms poll, content updates randomized between 200 ms and 5 s.

4.1.1.4 Multiple Clients

A new piece of content may match multiple clients subscriptions, so in addition to latency for a single content notification being sent to a single client, latency was measured for one new content notification delivery to multiple clients.

4.1.1.4.1 Latency

Latency for a single new content notification sent to multiple clients was compared for push and poll methods. For the push method, all clients would connect to the server then a new content notification would be sent to the server and subsequently sent to all connected clients. Latency was measured from the time of arrival of the new content notification to the server to the time of receipt of each client's acknowledgement message. For the poll method, at time $t = t_0$ all clients would start polling at an interval of 200 milliseconds. A new content notification message would be sent to the server at a random time between 200 milliseconds and 5 seconds after t_0. Latency was measured between the first poll message sent and the receipt of each client's acknowledgement message.

For 10 clients the mean latency was 36.6 milliseconds for push and 198.1 milliseconds for poll (Figure 4.5, Figure 4.6). The push method's latency is around 18.4% that of the poll method. For 100 clients the mean latency was 62.6 milliseconds for push and 350.6 milliseconds for poll (Figure 4.7, Figure 4.8). The push latency is only 17.9% for that of the poll method. For 1000 clients the mean latency was 1384.2 milliseconds for push and 7738.2 milliseconds for poll (Figure 4.9, Figure 4.10). This test also shows the push method's latency to be only 17.9% that of the poll method. However, for 1000 clients, only 353 actually retrieved a new content

notification. Inspection of the Apache Tomcat HTTP server log revealed that the other 647 clients' requests either timed out or were rejected because the incoming connection queue was full. Since the server was getting requests from 1000 clients at an interval of every 200 milliseconds, the maximum number of request processing threads was reached and additional requests were queued. A maximum numbers of threads from the internal thread pool can be used for request processing – this number is set based on the number of CPUs/cores the server has. Once this limit has been reached, subsequent requests are queued. The incoming connection queue has a maximum number of requests it can hold – this number is a tunable parameter which must be set according to the amount of traffic the server will receive, its network connection, as well as the server's system specifications i.e. number of CPUs/cores, CPU speed, and memory. Once that queue is full, further requests are rejected. When a large number of clients, e.g. 1000, poll the server frequently, e.g. every 200 milliseconds, the web server quickly gets flooded and cannot serve most requests. In contrast, the push method can serve 5000 clients on a single server without issue with an average content notification latency of 3094.3 milliseconds (Figure 4.11). The standard deviation of latency is relatively high compared to the mean due to the limited resources of a single machine i.e. maximum number of threads that can run concurrently.

The push method of new content notification is superior to the poll method because its latency is less than 20% that of the latter and can handle a larger number of simultaneous clients.

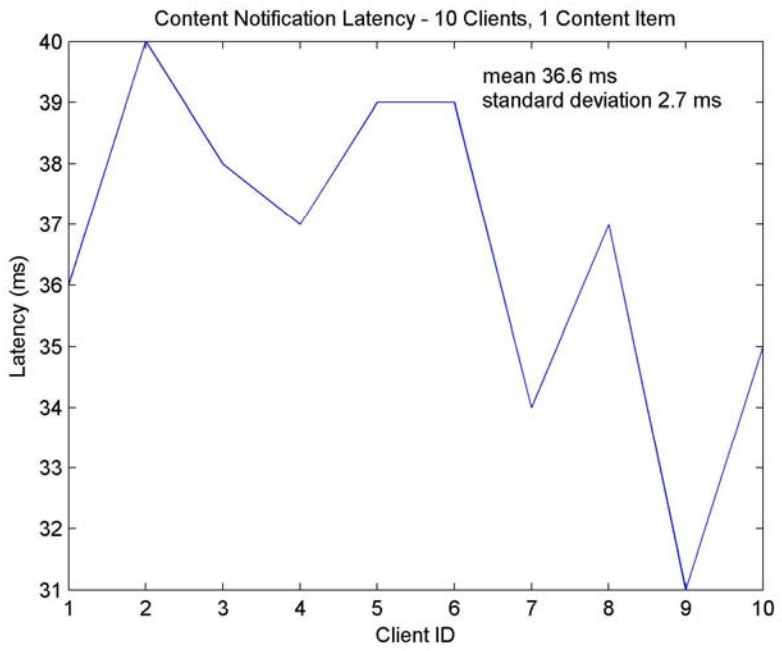

Figure 4.5 Content notification latency - push, 10 clients, 1 content item

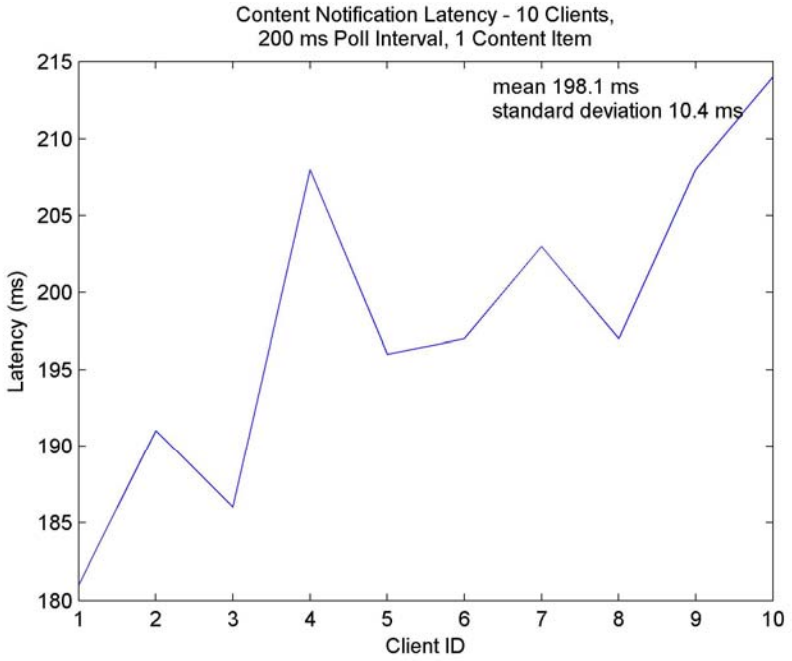

Figure 4.6 Content notification latency – 200 ms poll, 10 clients, 1 content item

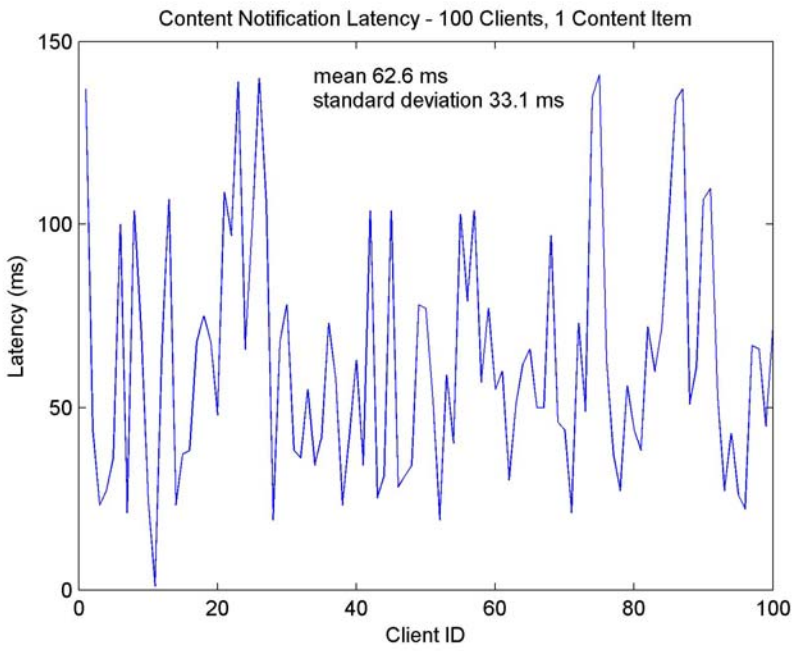

Figure 4.7 Content notification latency - push, 100 clients, 1 content item

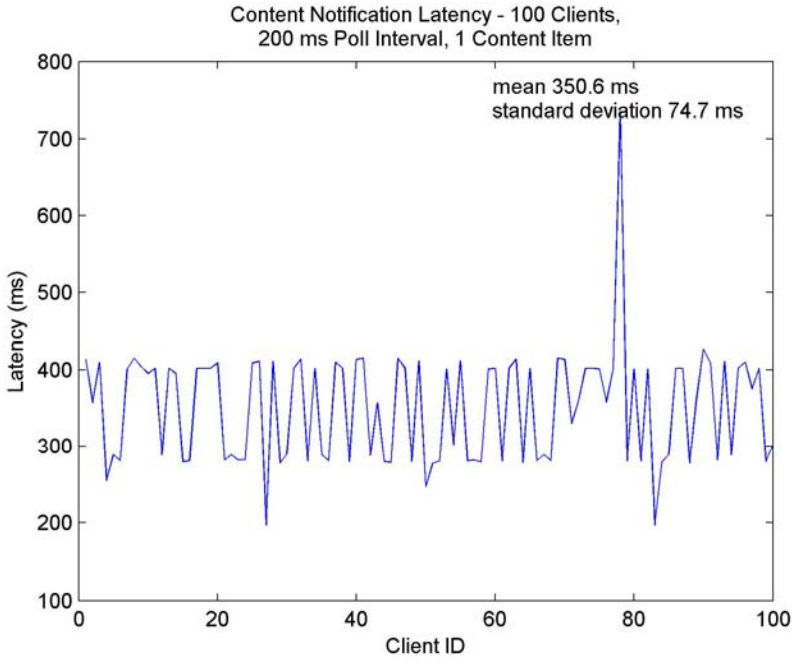

Figure 4.8 Content notification latency - 200 ms poll, 100 clients, 1 content item

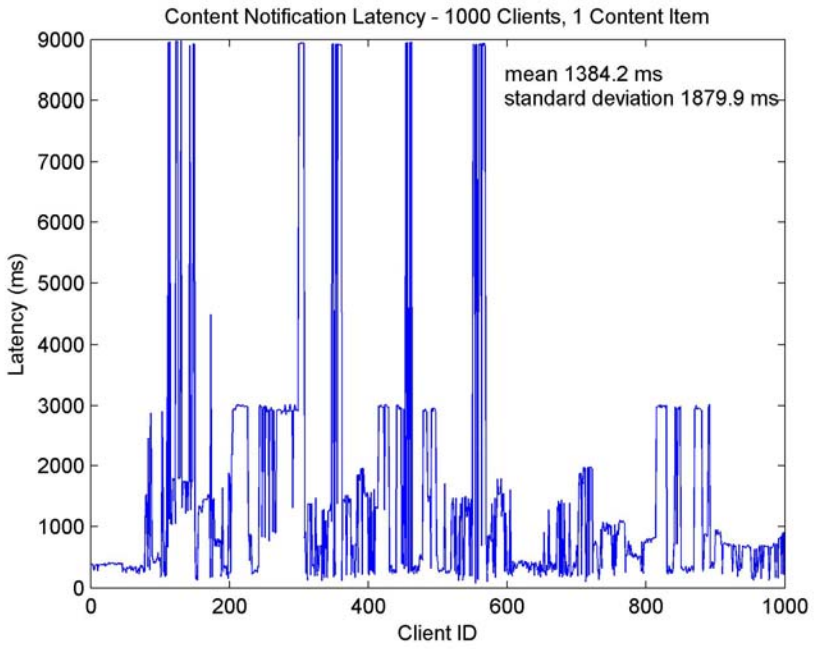

Figure 4.9 Content notification latency - push, 1000 clients, 1 content item

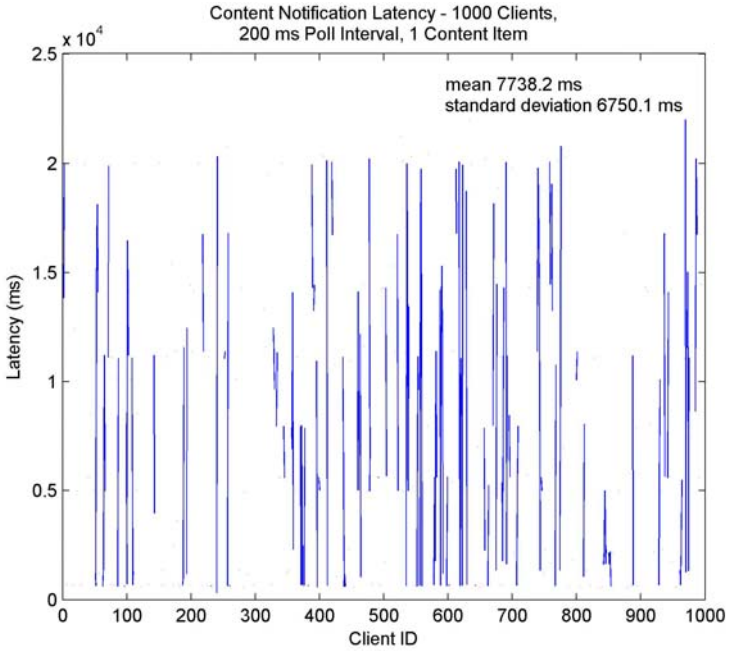

Figure 4.10 Content notification latency - 200 ms poll, 1000 clients, 1 content item

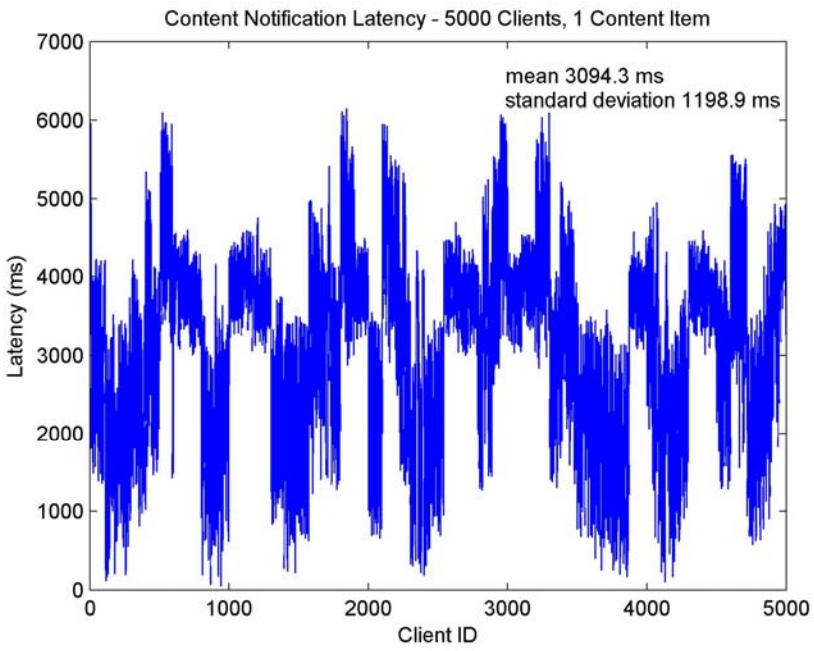

Figure 4.11 Content notification latency - push, 5000 clients, 1 content item

4.1.2 Rule Spatial Index

When new connect is sent to the middleware, every client's subscription Rules must be searched for a match. As the frequency of new content arrivals and/or the number of clients increases, the Rules search will take more time. In order to decrease the time needed to for matching Rules, a location indexing scheme for Rules was implemented. The performance of searching Rules using the location index versus performing a SQL search is measured.

4.1.2.1 Test setup

One client was connected with an active subscription comprised of one Rule. The Rule was comprised of one data type condition -- a location condition defined by an upper left latitude,

longitude coordinate of (34.068333, -118.444609) and lower right of (34.067778, -118.443375) (Figure 4.12). This location was indexed within the Tile with upper left and lower right coordinates of (35.0, -118.1) and (34.0, -118.0) (Figure 4.13). New content was added and search times were measured for index only, SQL only, and an Amazon relational database searches. Two scenarios were tested: 1) new content location lies within the Rule's bounding box 2) new content's location lies outside of the Rule's bounding box. The SQL server and the test client were on the same LAN and ping times between the two were less than 1 millisecond.

4.1.2.2 Search Results

The first scenario, in which new content was located within the Rule's location bounding box, yielded a mean index search time of 0.02 milliseconds and a mean SQL search time of 1.28 milliseconds over 100 trials (Figure 4.14). Cumulatively over 100 trials, the index search took 1.80 milliseconds and the SQL search took 127.90 milliseconds (Figure 4.15). The second scenario, in which new content was located outside the Rule's location bounding box, yielded a mean index search time of 0.01 milliseconds and a mean SQL search time of 1.11 milliseconds over 100 trials (Figure 4.16). Cumulatively over 100 trials, the index search took 0.52 milliseconds and the SQL search took 110.62 milliseconds (Figure 4.17). In both cases the index search time was smaller by two orders of magnitude. The index search is much faster than the SQL search because it is stored in memory and uses a constant sized array. Tile access is $O(1)$ i.e. constant time and R-Tree search time is on average $O(\log n)$. Searching the SQL database incurs the delay of traversing the network as well as the time taken to perform the actual search. Cumulatively the search time taken for SQL searches would significantly contribute to the delay in sending new content to clients. The Rules index minimizes the time taken to match new

content to user subscriptions and allows the middleware to deliver new content notifications in near real-time.

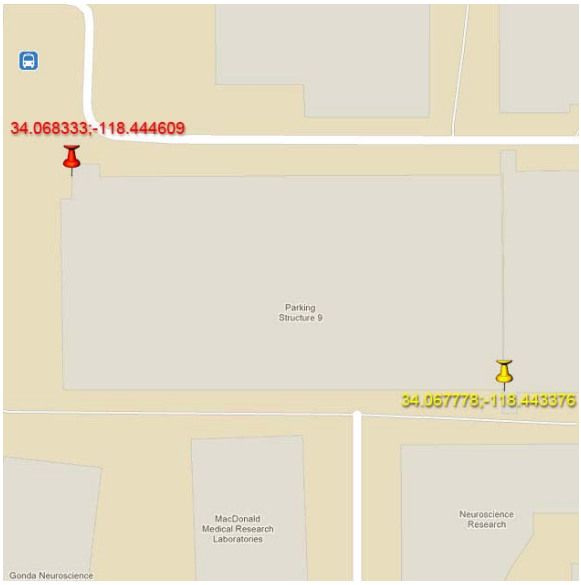

Figure 4.12 Upper left and lower right coordinates of Rule's location bounding rectangle.

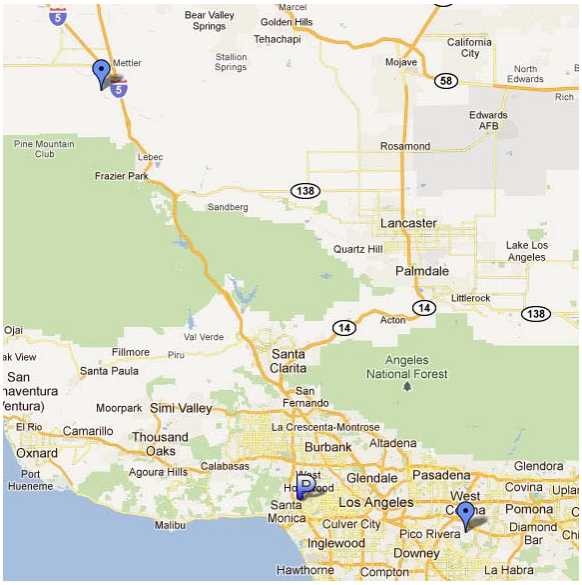

Figure 4.13 Tile upper left and lower right shown by blue pins. Parking lot 9 indicated by "P".

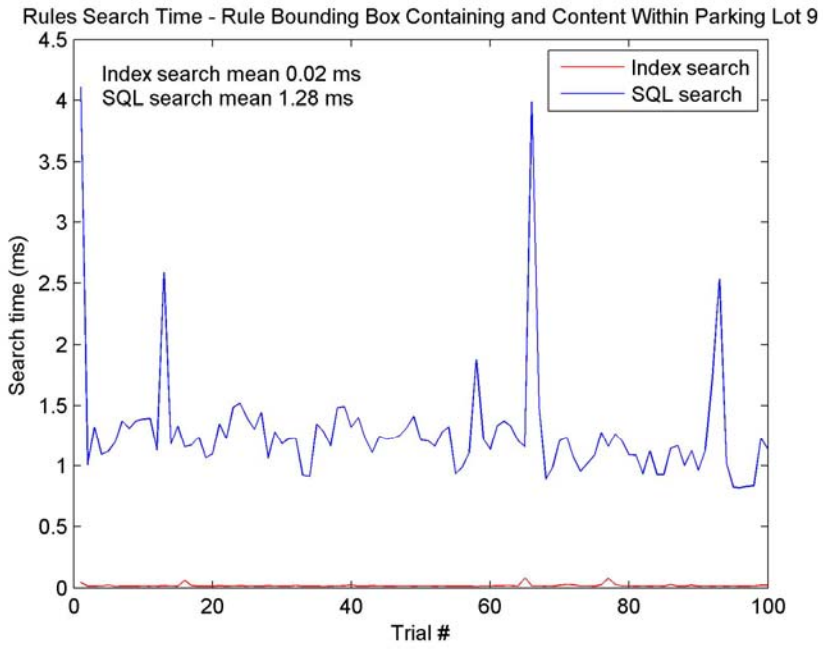

Figure 4.14 Rule search time - Rule bounding box contains and content lies within parking lot 9.

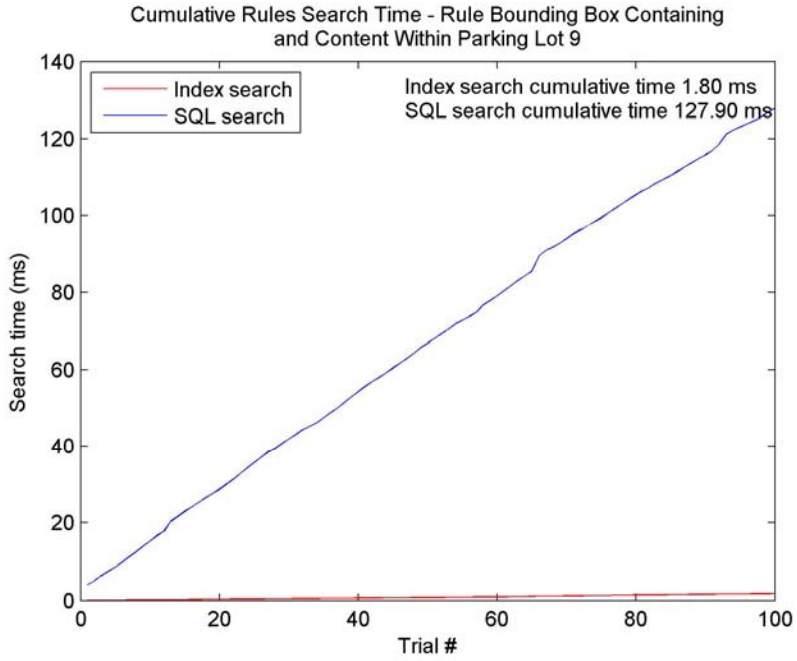

Figure 4.15 Cumulative Rule search time - Rule bounding box contains and content lies within parking lot 9.

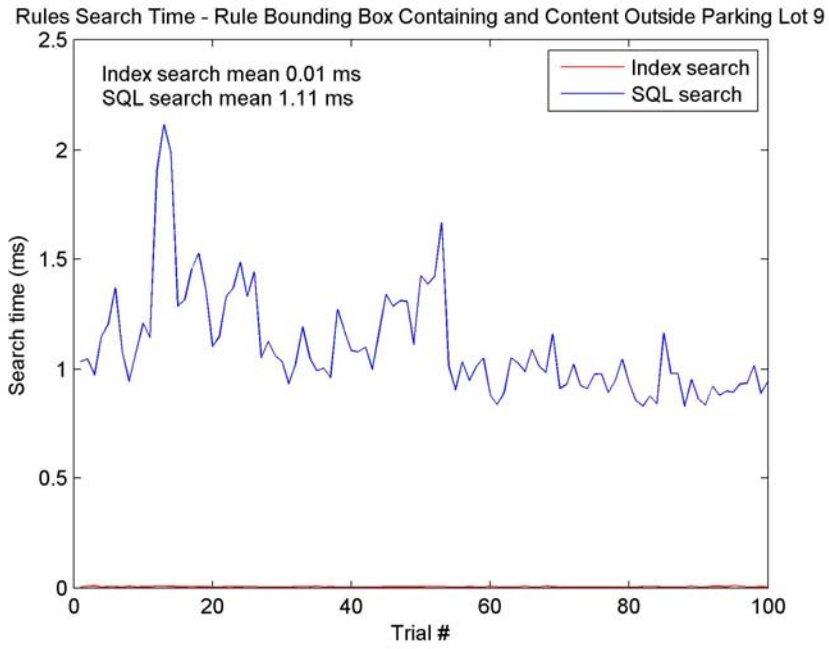

Figure 4.16 Rule search time - Rule bounding box contains and content lies outside of parking lot 9.

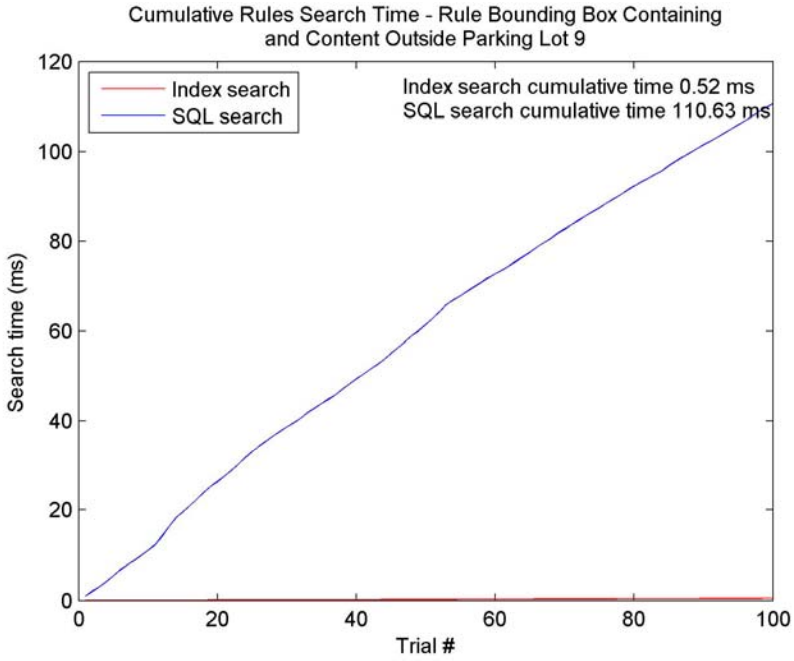

Figure 4.17 Cumulative Rule search time - Rule bounding box contains and content lies outside of parking lot 9.

4.1.3 Content Spatial Index

All content must be searched when: 1) a Rule is updated, either when a user explicitly changes his condition value parameters or the middleware automatically updates a Rule's "My Location" condition based on a device context update 2) a new Rule is created or an old Rule is activated as a result of a user connecting. The accuracy of the content spatial index (CSI) in indexing content location is demonstrated. Search performance of the CSI versus various indexed and non-indexed SQL searches is measured.

4.1.3.1 Test setup

4.1.3.1.1 MAXCONTENT

Search times were measured for different *MAXCONTENT* values to determine an optimal value. A single Tile with uniformly distributed content was searched using a search rectangle defined by an upper left latitude, longitude coordinate of (0.2, 0.1) and lower right of (0.1, 0.2).

4.1.3.1.2 Sub-Tiling

Content items with a uniform location distribution are added to a Tile to ensure that sub-Tile creation for uniformly distributed content works. Three clusters of content are added to a Tile with uniform location distribution to ensure that sub-Tile creation at clusters functions when both clustered and uniform content are present.

4.1.3.1.3 Search Times

Searches of 1, 30, 120, 280, and 500 km^2 are performed and search times are measured for the two previously mentioned location distributions as well as of a single Tile and an empty Tile.

The test Tile is defined by an upper left latitude, longitude coordinate of (1.0, 0.0) and lower right of (0.0, 1.0). The Tile search times are compared with two traditional SQL searches: 1) a SQL indexed spatial geography search 2) a SQL indexed latitude and longitude value search. Search times for a cloud database are measured during morning and evening as a baseline comparison for Tile searches.

4.1.3.2 Results

4.1.3.2.1 MAXCONTENT

MAXCONTENT values were varied from 100 to 100,000 and searches were performed (Figure 4.18). A search time less than one millisecond was desired, so it was determined that 1000 was the optimal balance between content capacity and search time for *MAXCONTENT*.

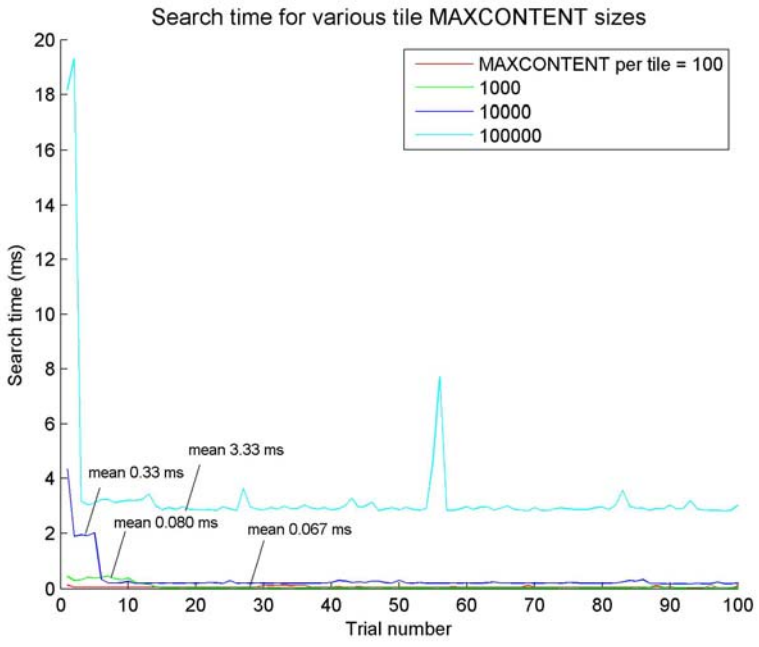

Figure 4.18 MAXCONTENT search times.

4.1.3.2.2 Sub-Tiling

Content items with uniform location distribution are uploaded to the middleware. The CSI indexes them and they are added to a Tile. The calculated *p*-value is 0.99 which signifies that the content is spatially uniformly distributed. The "create n equal sub-Tiles" function is called and the content containing Tile is divided into four equal sub-Tiles, each containing a subset of their parent Tile's content (Figure 4.19). Next, a combination of uniformly distributed and clustered content is uploaded to the middleware and indexed by the CSI. The calculated *p*-value is 0.42 which signifies that the content is spatially clustered. The "create sub-Tiles at clusters" function is called and sub-Tiles are created that contain each of the three clusters of content (Figure 4.20).

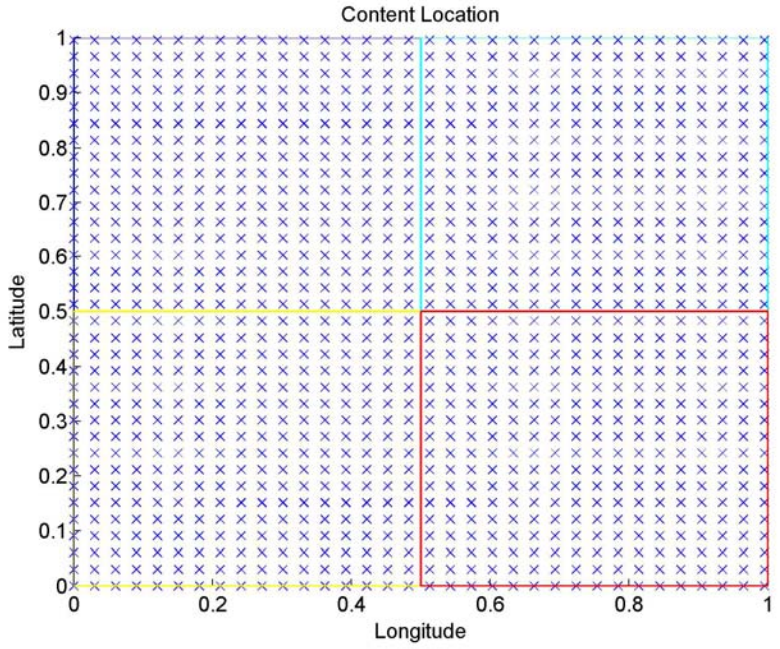

Figure 4.19 Spatially uniformly distributed Tile divided into four equal sub-Tiles by the "create n equal sub-Tiles" function.

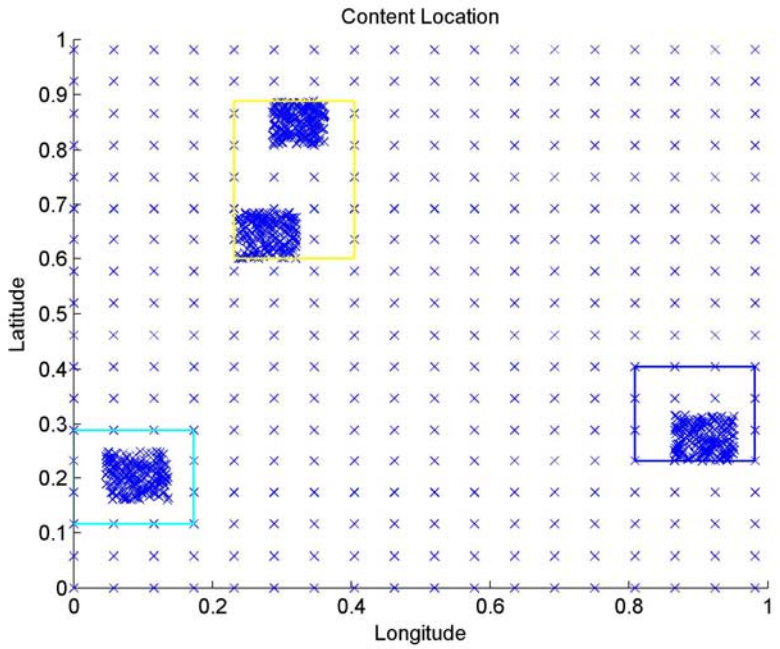

Figure 4.20 Spatially clustered and uniform content. Three sub-Tiles created by the " create sub-Tiles at clusters " function.

4.1.3.2.3 Search Times

The first test scenario was a single Tile with the maximum number of content items (1000) and varied search region sizes. Tile search times were compared to those of a SQL indexed spatial geography search and a SQL indexed latitude and longitude value search (Figure 4.21). It is shown that over all search area sizes the Tile search takes a mean time of 0.18 milliseconds, while the SQL lat/lon search takes 23.6 milliseconds, and the SQL geography search takes 259.1 milliseconds (Figure 4.22, Figure 4.23, Figure 4.24, Figure 4.25, Figure 4.26).

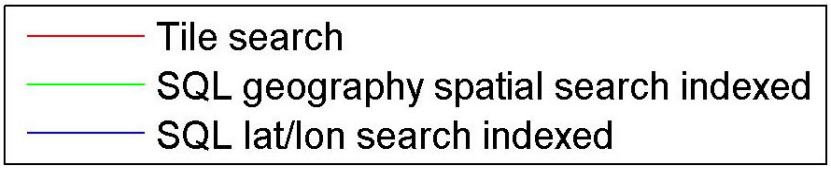

Figure 4.21 Search type legend.

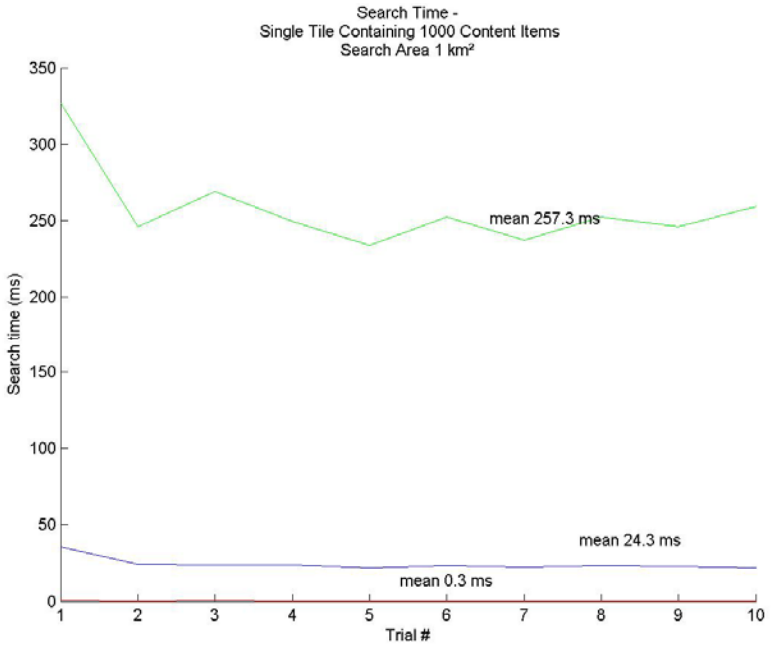

Figure 4.22 Single Tile search times - 1 km^2 area.

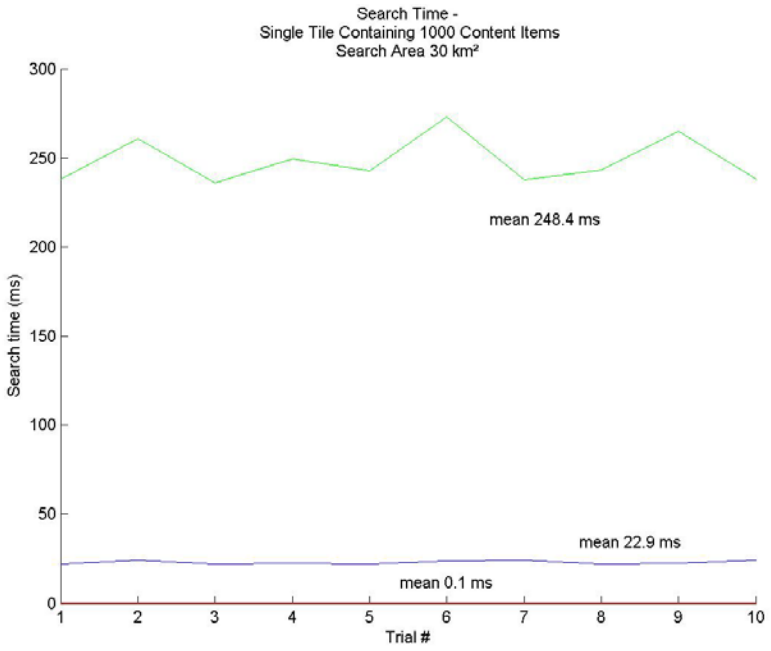

Figure 4.23 Single Tile search times - 30 km^2 area.

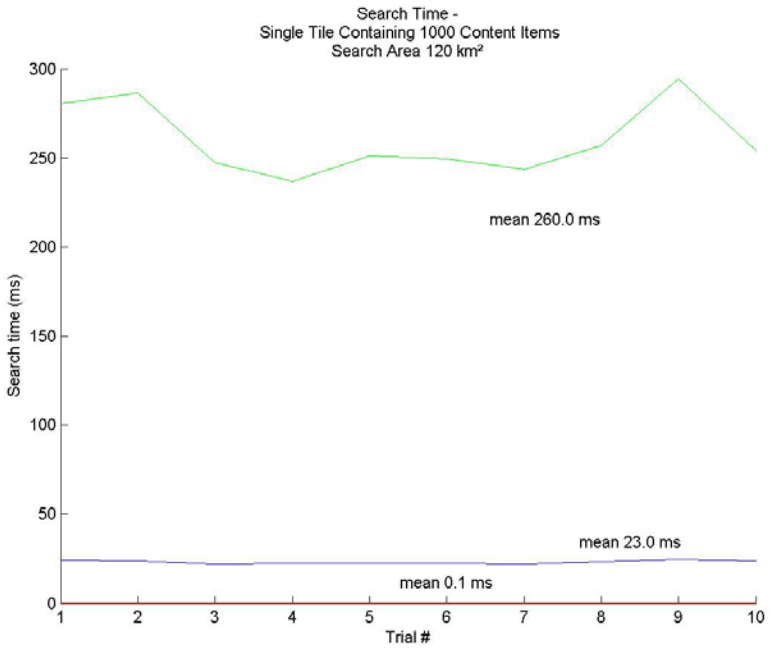

Figure 4.24 Single Tile search times - 120 km^2 area.

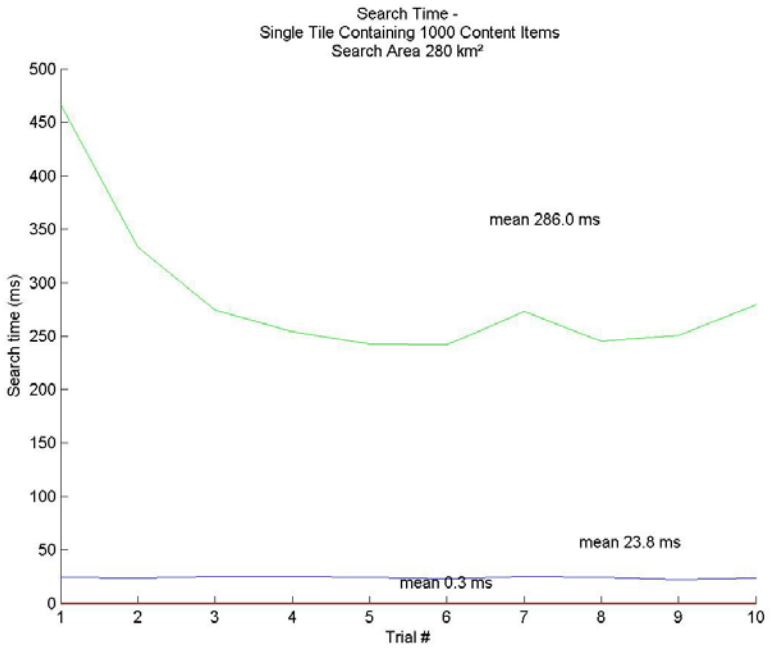

Figure 4.25 Single Tile search times - 280 km^2 area.

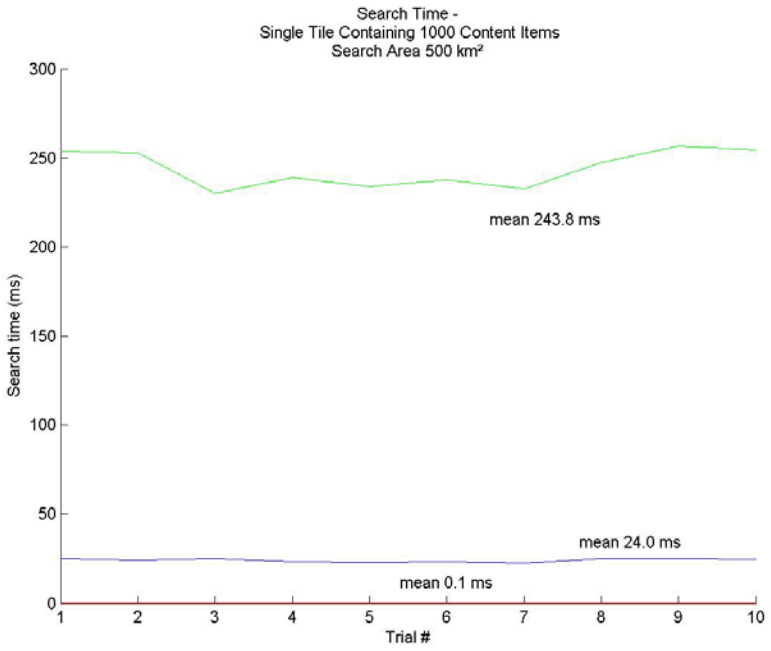

Figure 4.26 Single Tile search times - 500 km^2 area.

The second test scenario was an empty Tile with the maximum number of content items (1000) and varied search region sizes. The Tile search time was recorded as 0.0 milliseconds for most of the searches because the precision of the timing function was limited to 0.1 milliseconds. The first search of 1 km^2 area yielded a mean search time of 0.4 milliseconds because the first index query is much slower due to incurring a cache miss (Figure 4.27). The mean time over all tests for SQL lat/lon searches was 23.4 milliseconds, and 254.7 for SQL geography searches (Figure 4.27, Figure 4.28, Figure 4.29, Figure 4.30, Figure 4.31).

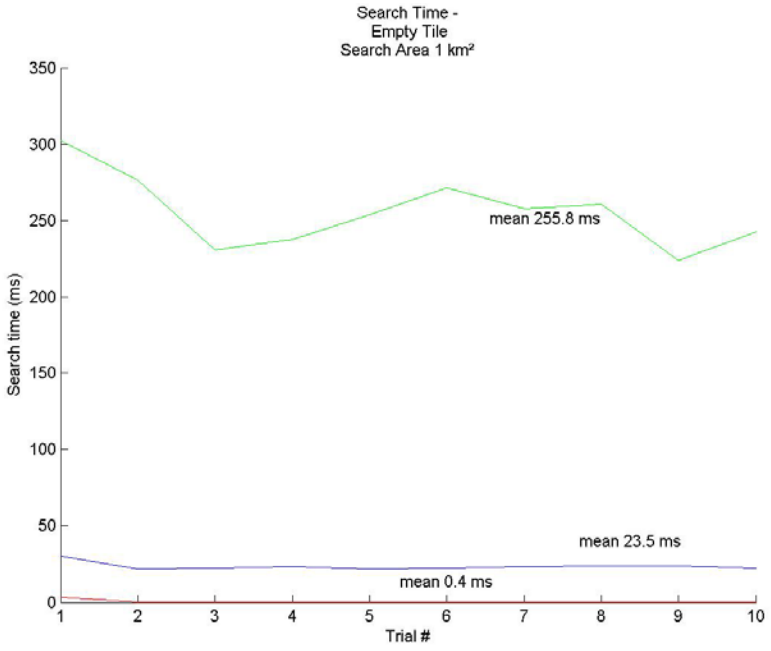

Figure 4.27 Empty Tile search times - 1 km^2 area.

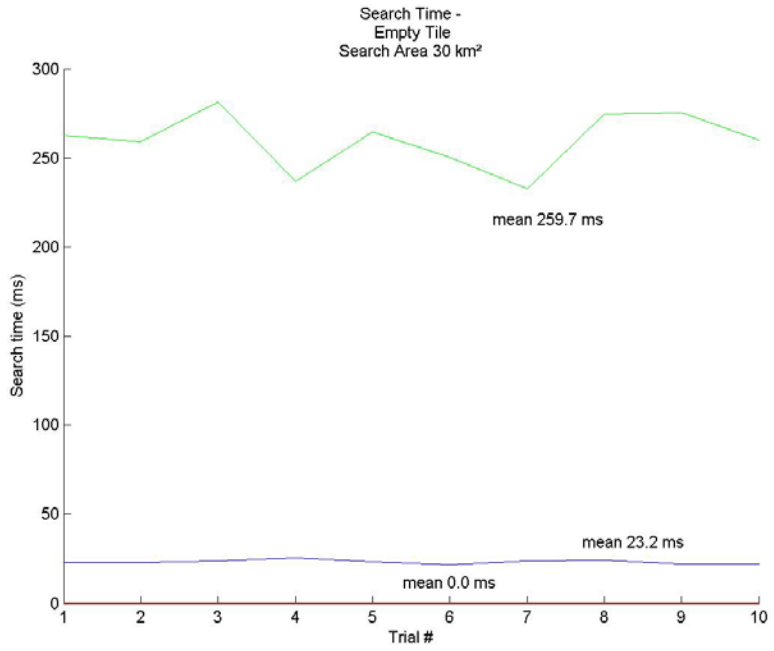

Figure 4.28 Empty Tile search times - 30 km^2 area.

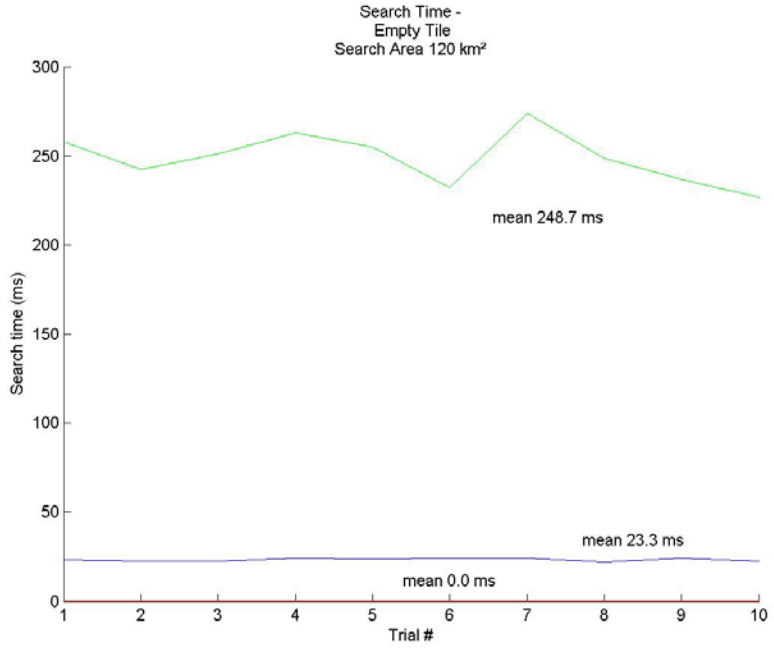

Figure 4.29 Empty Tile search times - 120 km^2 area.

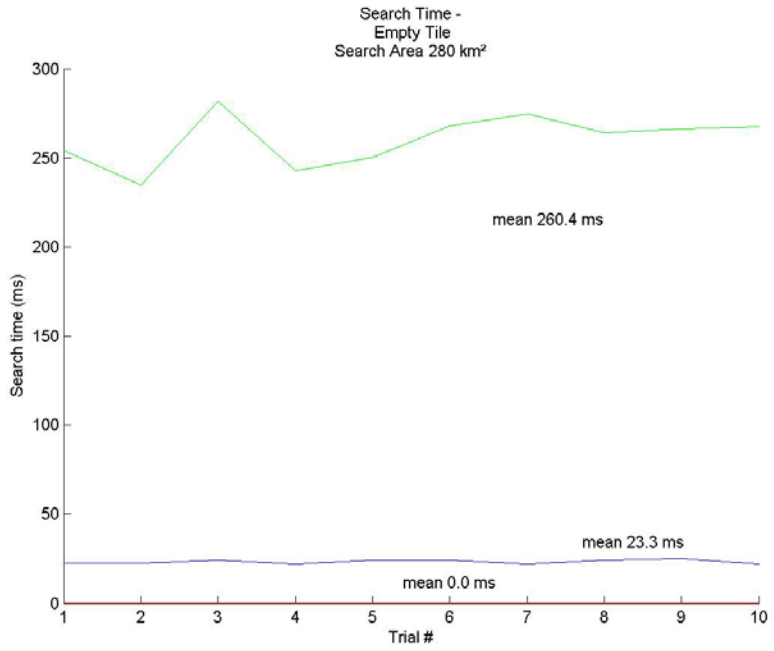

Figure 4.30 Empty Tile search times - 280 km^2 area.

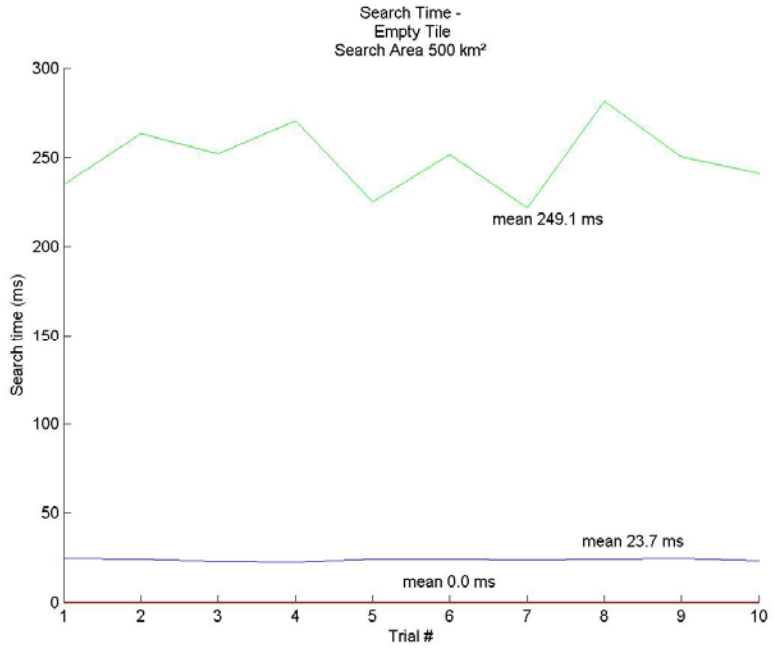

Figure 4.31 Empty Tile search times - 500 km^2 area.

The third test scenario was a single Tile with four equally sized sub-Tiles each containing 250 content items. The following search regions, denoted by upper left and lower right lat/lon pair, were used: 1) (0.750000,0.750000), (0.740000,0.760000) 2) (0.750000,0.750000), (0.700000,0.800000) 3) (0.750000,0.750000), (0.650000,0.850000) 4) (0.750000,0.750000), (0.600000,0.900000) 5) (0.750000,0.750000), (0.550000,0.950000). The Tile search time averaged 0.12 milliseconds over all search area sizes. The mean time over all tests for SQL lat/lon searches was 23.7 milliseconds, and 264.1 for SQL geography searches (Figure 4.32, Figure 4.33, Figure 4.34, Figure 4.35, Figure 4.36).

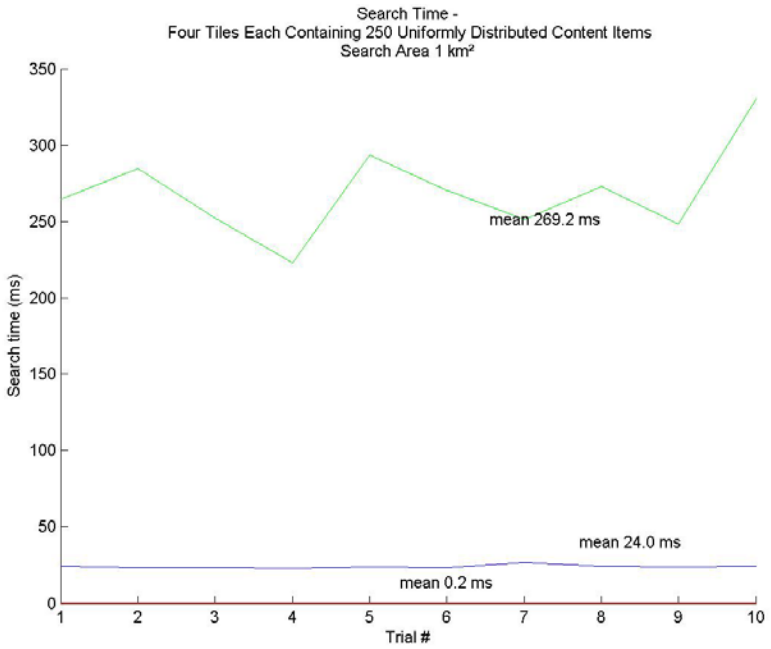

Figure 4.32 Four sub-Tiles search times - 1 km^2 area.

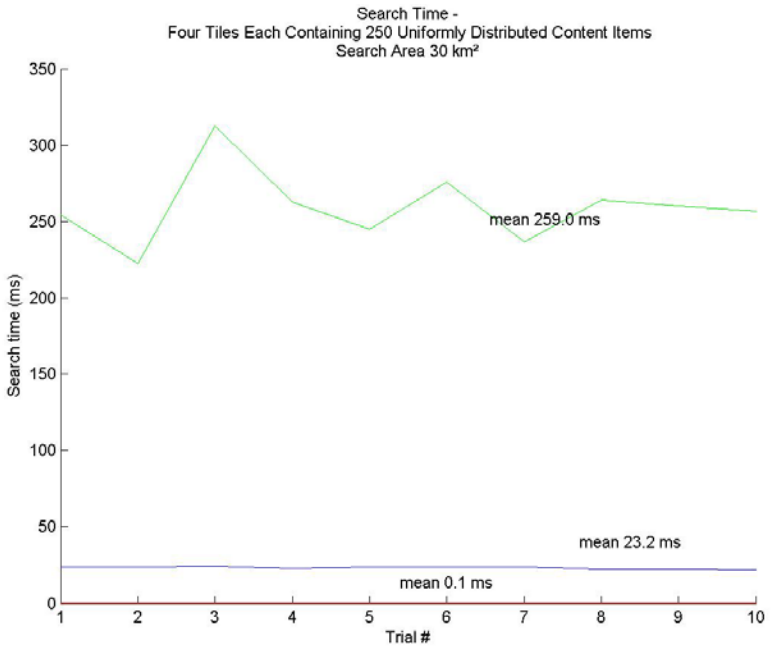

Figure 4.33 Four sub-Tiles search times - 30 km^2 area.

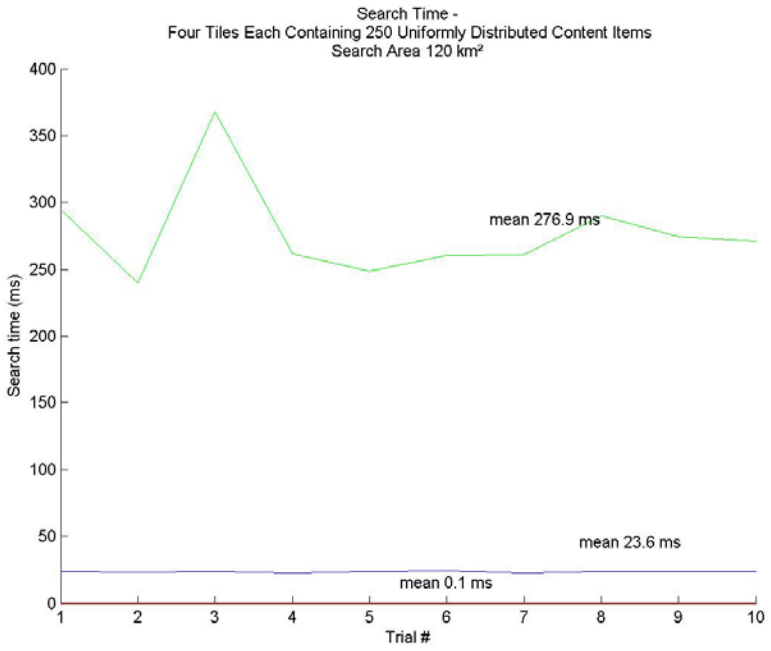

Figure 4.34 Four sub-Tiles search times - 120 km^2 area.

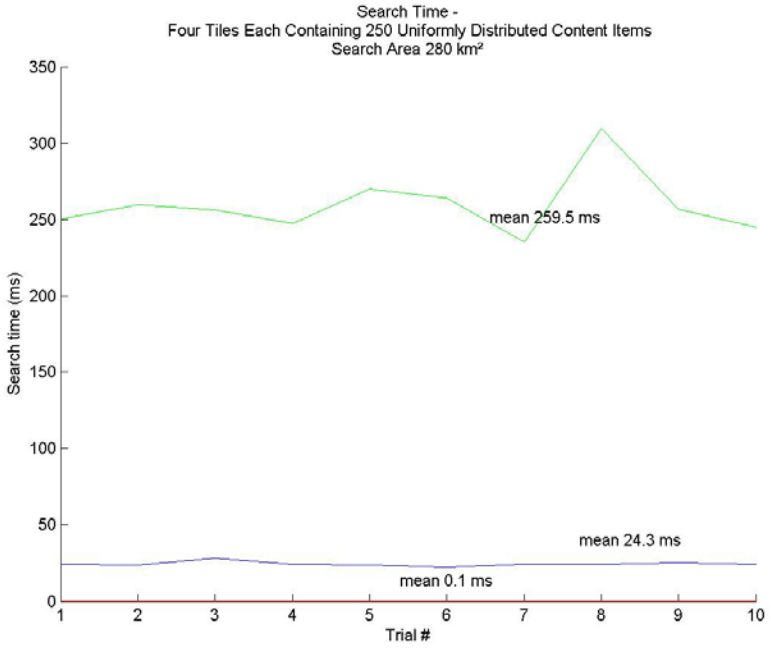

Figure 4.35 Four sub-Tiles search times - 280 km^2 area.

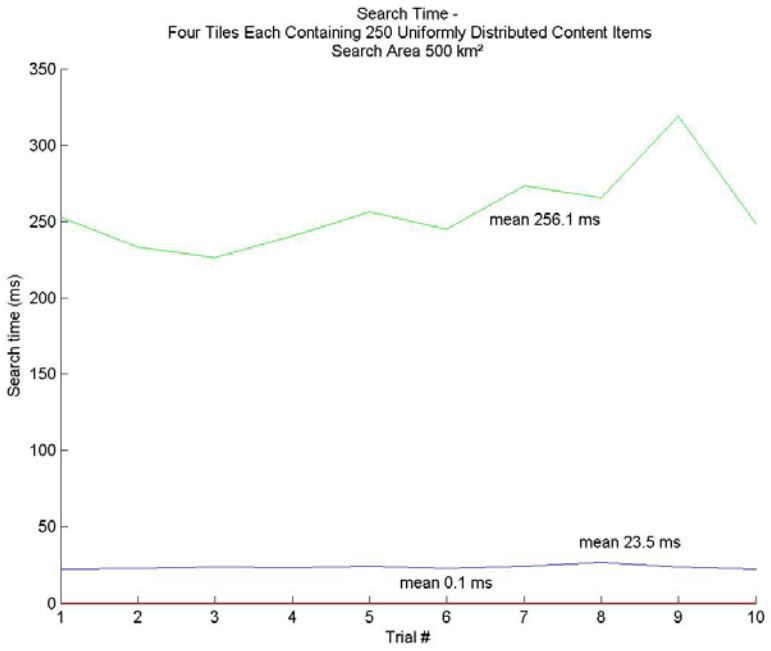

Figure 4.36 Four sub-Tiles search times - 500 km^2 area.

The fourth test scenario was a single Tile containing uniformly distributed content and three sub-Tiles containing clustered content. Two searches were performed, one within the main Tile and one within a cluster sub-Tile. The Tile search time averaged 0.05 milliseconds over both search areas. The mean time over all tests for SQL lat/lon searches was 22.7 milliseconds, and 244.6 for SQL geography searches (Figure 4.37, Figure 4.38).

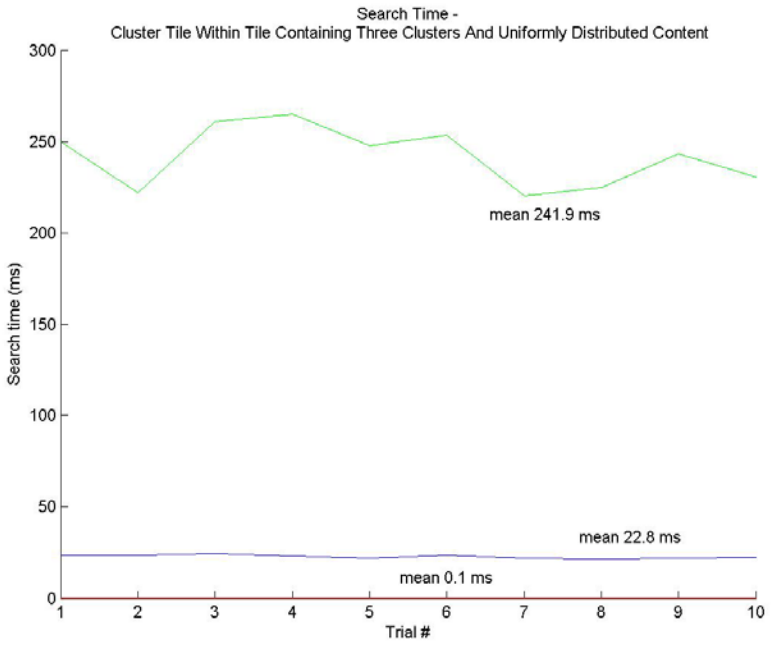

Figure 4.37 Cluster Tile search times.

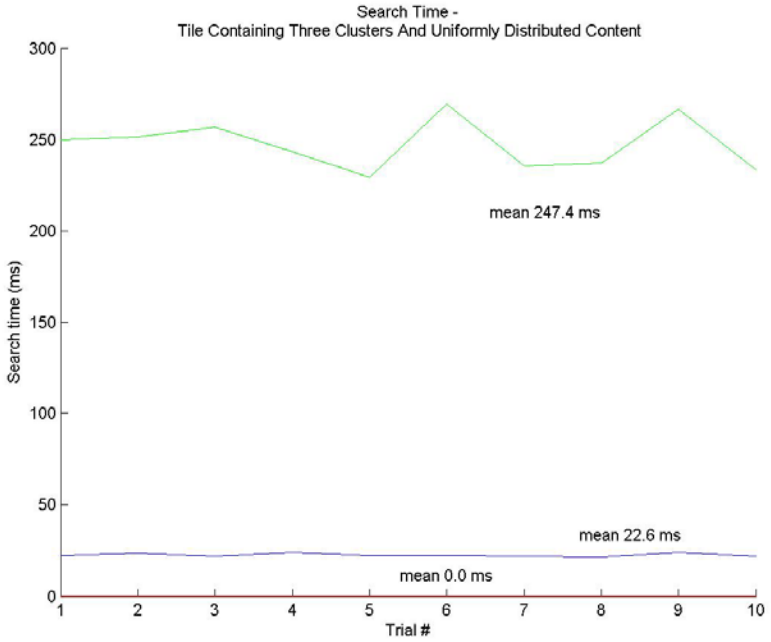

Figure 4.38 Main Tile search times.

An area bounded by (0.750000,0.750000), (0.740000,0.760000) was searched using an Amazon SimpleDB cloud database over 10 trials at 0900 PST and 0100 PST. The Tile search took 0.1 milliseconds for both trials. The cloud-based SimpleDB search took an average of 52.7 milliseconds over 10 trials at 0100 PST (Figure 4.40), a time of minimal Internet traffic, and 429.3 milliseconds at 0900 PST, a time of heavy Internet traffic (Figure 4.39).

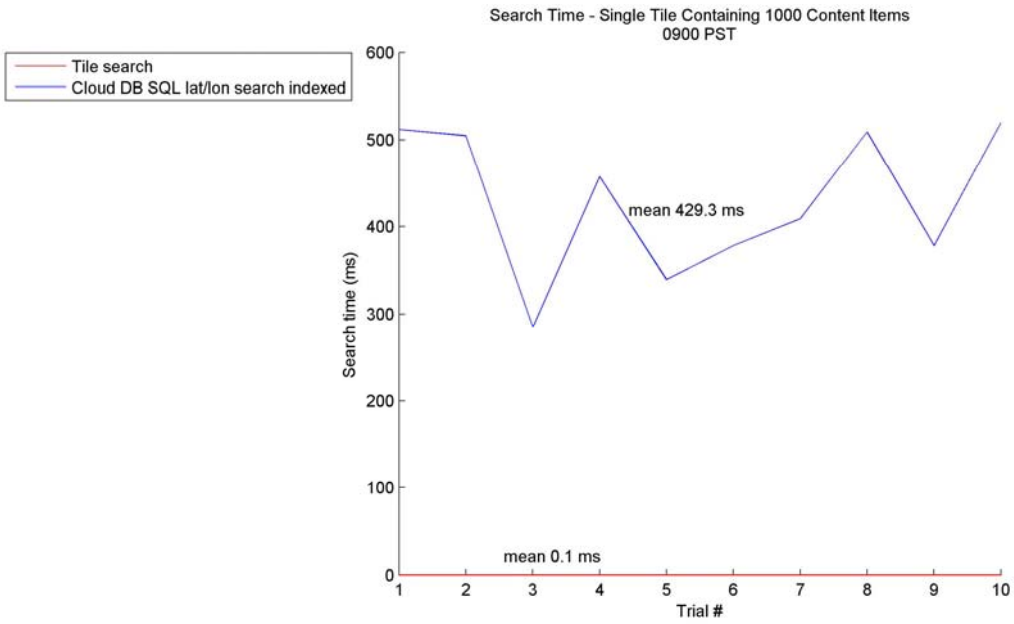

Figure 4.39 Cloud database search times 0900 PST.

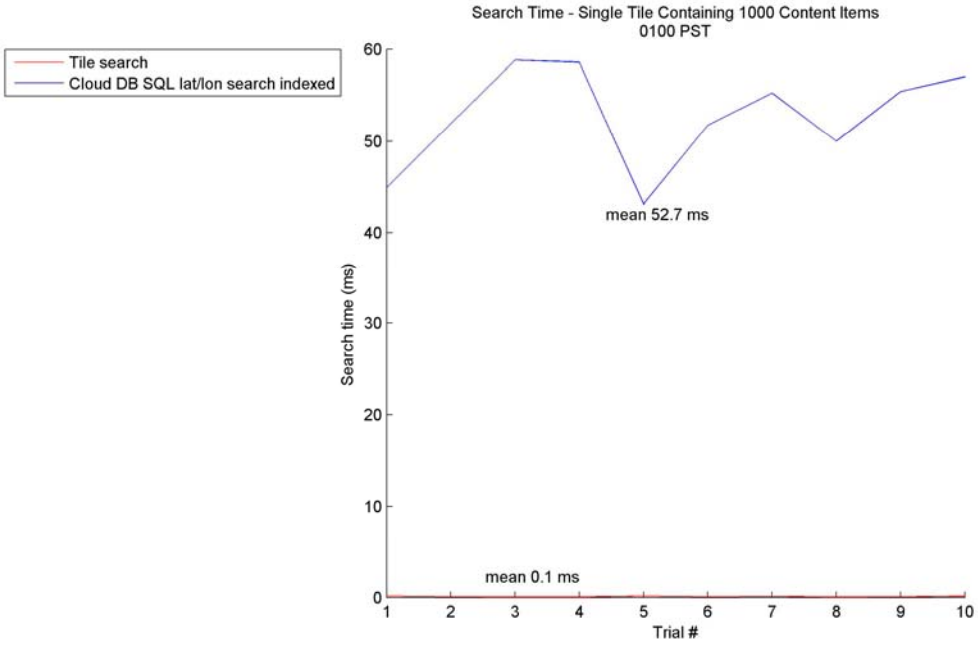

Figure 4.40 Cloud database search times 0100 PST.

4.1.3.3 Summary

The CSI search method took less than one millisecond for searching uniform and clustered content spatial distributions – two orders of magnitude less than SQL indexed lat/lon searches and three orders of magnitude less than SQL indexed geography searches. The advantage of an in-memory index is further magnified when compared to a cloud database search, where search times varied from 52.7 milliseconds to 429.3 depending on the broader Internet's traffic which varies according to the time of day. CSI searches are consistently faster than indexed SQL searches for location based content.

4.2 Connection Management

The functionality of the adaptive connection manager (ACM) is verified by traversing WiFi and 3G networks and moving through space at different velocities. Both tests started inside the Engineering 4 building where the iPhone device was connected via WiFi to the middleware. In one test a car was driven from UCLA to Santa Monica where the test ended (Figure 4.42). In the other test the device was carried while walking from Engineering 4 in a loop around campus then returning to Engineering 4 (Figure 4.44). In the driving test, the ACM had to reconnect a total of three times: the first and last times were caused by transitioning from WiFi to 3G and the second time was due to a loss in 3G data connectivity. Content notification latency and velocity are plotted versus time (Figure 4.41). Every content notification arrived successfully, with the longest latency being 1500 milliseconds. In the walking test, the ACM had to reconnect twice, both when transitioning from WiFi to 3G and back. A total of four notifications did not arrive and had to be resent during the walking trial – shown by the four spikes extending beyond the *y*-limit between 0 and 300 seconds (Figure 4.43). This is due to being in areas of limited to no 3G

connectivity such as inside an elevator or stairwell. The ACM is shown to satisfactorily manage the socket connection across networks.

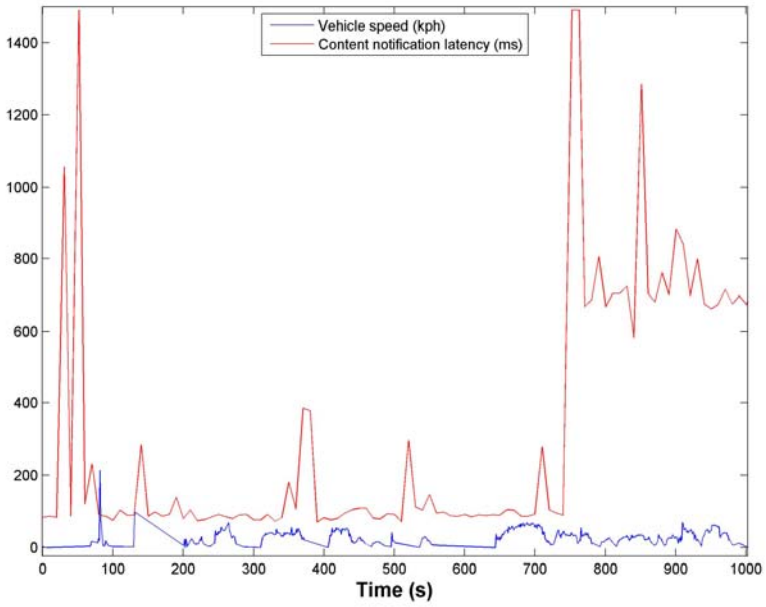

Figure 4.41 Content notification, velocity versus time - driving.

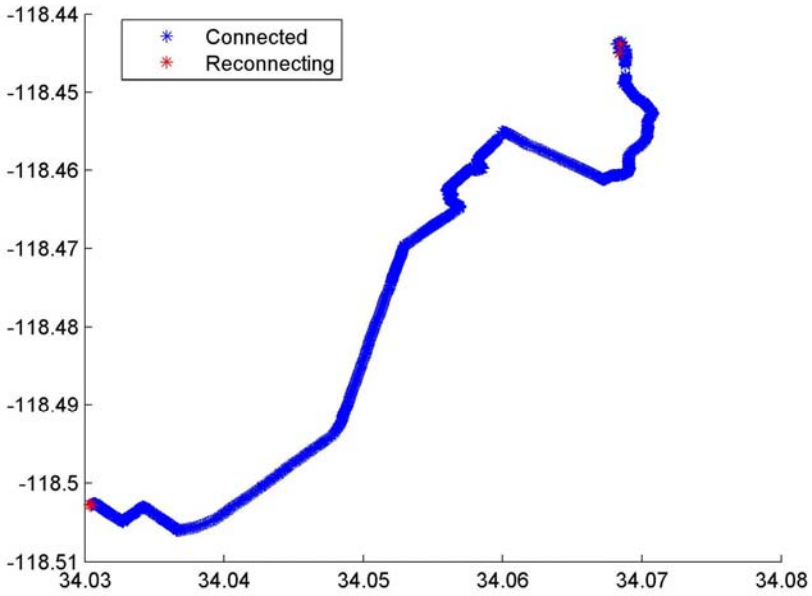

Figure 4.42 Spatial location - driving.

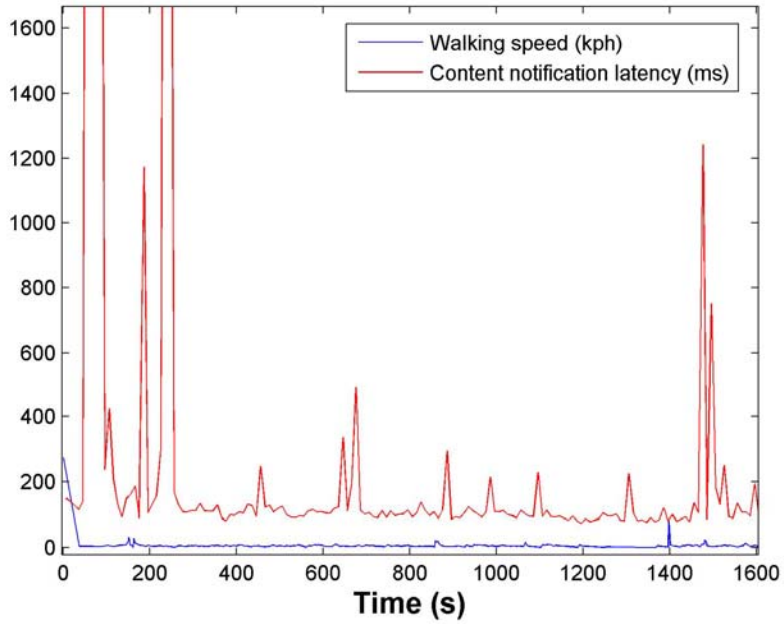

Figure 4.43 Content notification, velocity versus time - walking.

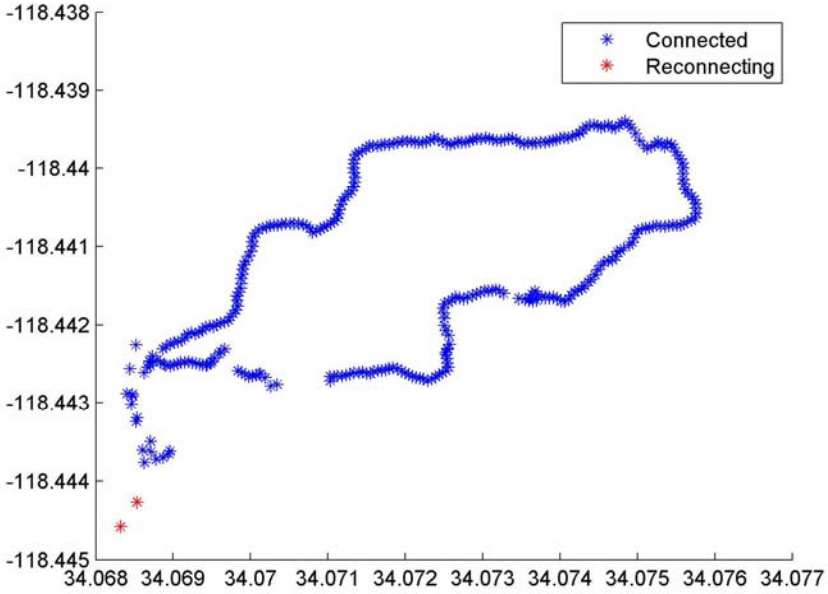

Figure 4.44 Spatial location - walking.

Chapter 5 Pilot Studies

The NRTPM was implemented as part of a near real-time content distribution platform for mobile devices targeted at college sporting events called MobiSportsLive. In collaboration with Motorola, pilot studies comparing the TuVista application with MobiSportsLive were conducted. System performance in delivering real-time photos and videos was measured.

5.1 TuVista

TuVista is a content editing, storage, and distribution platform for mobile internet devices created by Motorola Research. It provides a human-in-the-loop solution for real-time content capturing, editing, encoding and distribution for mobile devices

5.1.1 System Architecture

The TuVista system is comprised of multiple discrete components that handle specific tasks (Figure 5.1). A live video feed is sent to a stream encoder which takes a video signal (e.g. composite, HDMI) and digitizes and encodes a video stream. The video data is compressed using the H.264 codec, audio using the MP3 codec, and both are contained in an ASF file. The bitstream is served over Real-time Transport Protocol (RTP). The video clip authoring workstation acts as a client and receives the streaming video from the server using Real-Time Streaming Protocol (RTSP). A person must watch the live stream and create clips using the authoring software. Clips are created by specifying a start and end time as well as descriptive tags. The created video clips are sent to a content transcoder that converts them to formats compatible with supported target mobile devices. Content resolution, codec, and bitrate are

converted based on the requirements of target devices. The transcoded files are stored on a content host and the TuVista application server is notified of their availability and URL. Mobile clients periodically poll the TuVista application server for new content via HTTP message. If content exists that is newer than the message's "Last-Modified" value, they are notified in the server's response of the new content's thumbnail, URL, and tags. This information is displayed to the user who may choose to view the content.

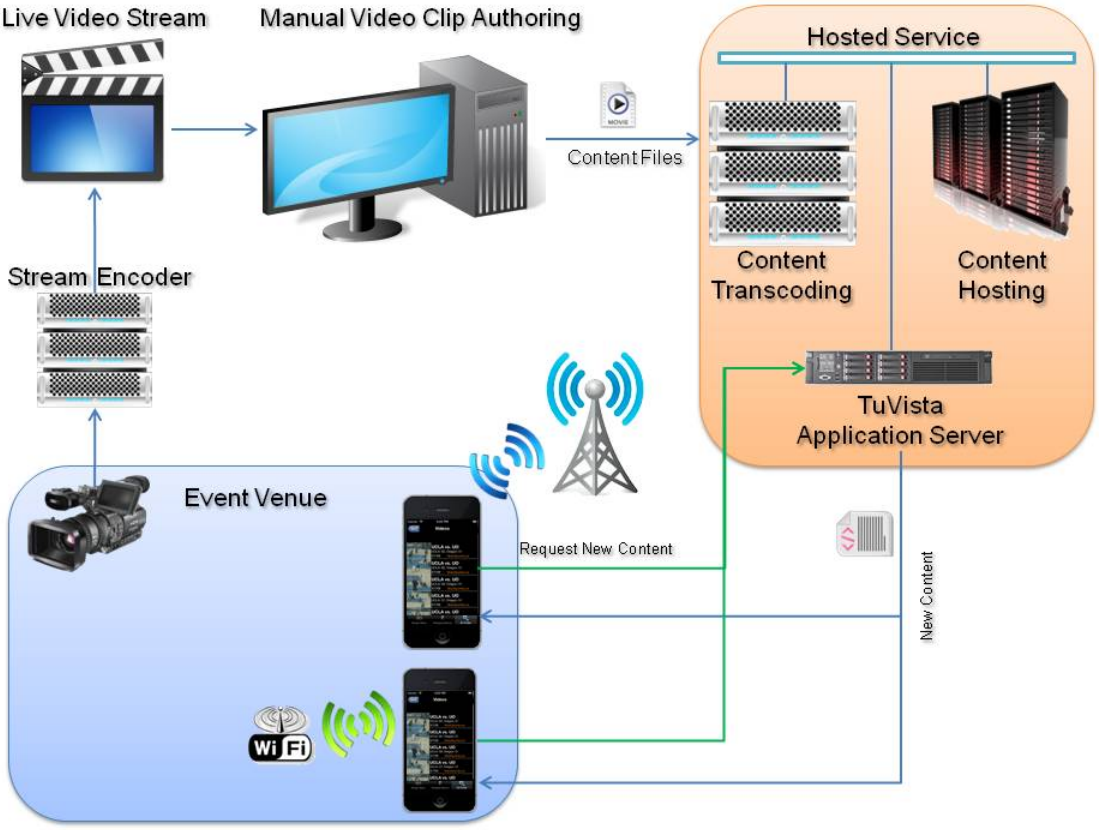

Figure 5.1 TuVista System Architecture

5.1.2 Logical Architecture

In addition to the system architecture comprised of physical hardware, the TuVista logical architecture is defined by four layers: 1) presentation 2) service 3) persistence 4) distribution. They are segregated by the type of functions they perform.

5.1.2.1 Presentation Layer

The Presentation Layer is comprised of all portals for human-computer interaction, including the mobile client portal, the administration portal, and the clip authoring portal.

5.1.2.1.1 Mobile Client Portal

The mobile client portal is the TuVista mobile application which provides functionality for user authentication, event selection, and new content selection and viewing. There are three views the user is presented with: 1) user authentication 2) event selection 3) new content notification. Once the user has entered his login credentials, he is presented with a list of all current and past events that he is authorized to view. When a past event is selected, all content published during that event is displayed. When a live event is selected, all published content is displayed and new content appears as it becomes available from the server. The mobile client polls the server for new content at an interval of one second.

5.1.2.1.2 Administration Portal

The administration portal allows an administrating entity to monitor and control individual system components. The performance and status of individual processes can be monitored. In addition, stalled or crashed processes may be restarted. User accounts and access are defined in

this portal as well. Groups may be created and users may be added to these groups. Content access can be limited by group or by individual user.

5.1.2.1.3 Clip Authoring Portal

The clip authoring portal application displays the real-time video stream and allows the user to create video clips. The incoming stream is cached to disk for clip creation and future access. The user selects the starting and ending time, relative to the beginning of the stream, for a clip. Once the user has selected the clip duration, the appropriate start and end time are sent to a separate clip encoding process. The clip encoder extracts the specified clip using the start and end time and encodes the clip using the H.264 codec, which is playable by most mobile devices. The clip and the user's descriptive tags are uploaded to the hosted service.

5.1.2.2 Service Layer

The Service Layer includes processes that directly support the portals of the presentation layer. The processes are user management, content management, and content creation.

5.1.2.2.1 User Management

User account and access management functionality is handled in this layer. Login and authentication occur securely over HTTPS. Wrapper functions handle database access and updating of user profiles and content viewing statistics.

5.1.2.2.2 Content Management

Content management encompasses distribution rights and constraints, and content view statistics. Content from a given event will be restricted to viewing by users authorized for that event. If the

URI of a piece of content is discovered by an outside party by sniffing or hacking, the content must be protected. To this end, user authentication is performed by a servlet each time content is accessed, so even if a content item's URI is known by an outside party, it will be inaccessible to them. In addition, the servlet tracks who and how many times content is accessed.

5.1.2.2.3 Content Creation

Content creation requires the capability to cache streaming content and split and re-encode the cached file. This is accomplished by helper functions and FFmpeg, a separate command-line program. Start and end time relative to the beginning of the streaming video and codec type are used to create each clip. The H264 is used because it is playable by most mobile devices.

5.1.2.3 Persistence Layer

The Persistence Layer provides interfaces to the various databases and flat files in which content, user data, and system data are stored.

5.1.2.4 Distribution Layer

The Distribution Layer encompasses all functionality needed to distribute content from the provider to the consumer. After a clip has been created, transcoded, and hosted, the TuVista application server makes the content available to clients. Clients poll the application server via HTTP request which responds with the latest content for the events for which they are authorized.

5.1.3 System Trial

5.1.3.1 Trial Setup

The TuVista system was deployed for a pilot study to measure real-world performance. The venue was UCLA's Pauley Pavilion and the event was a women's volleyball game. A total of 21 students, researchers, and Motorola staff used iPhones and iPod Touches as client devices. Of the 21 devices, 10 were connected over 3G and 11 were connected over WiFi. The WiFi users were spatially distributed throughout the venue so no more than 3-4 users were connected to one access point. The clip authoring workstation was located in the event venue and was connected via Ethernet. Since only one type of device was being supported (iPhone/iPod Touch), a separate transcoding server was unnecessary. Video was directly encoded on the clip authoring workstation for supported devices in two bitrates: 100 kbps and 300 kbps. The content store and TuVista application server were located in the WINMEC lab.

5.1.3.2 System Performance

Clip creation using the clip authoring portal took an average of 35 seconds, including user time taken by the user to set the clip's starting and ending time, enter tag information, as well as time to extract the video clip from the cached stream. Most clips created were 20 seconds long and transcoding took an average of 40 seconds to process both bitrate versions on a dual Intel Xeon 3.0 GHz with 1 GB of RAM. Transfer time from the clip authoring station to the content store took 50-150 milliseconds – uplink speed was 50 Mbps, and clip sizes were approximately 1 megabyte for 300 kbps quality and 350 kilobytes for 100 kbps quality. The TuVista application server took less than 1 second to add the content to the database and make it available to clients. The total time to create a clip inclusive of user time, transcoding time, transfer time, and server

application time, averaged 77 seconds. Clients poll the server every second to receive notification of the new content. A total of 87 clips were created and distributed to a variable number of clients. Clients arrived and departed throughout the duration of the game with a maximum of 21 and a minimum of 7 (Figure 5.2).

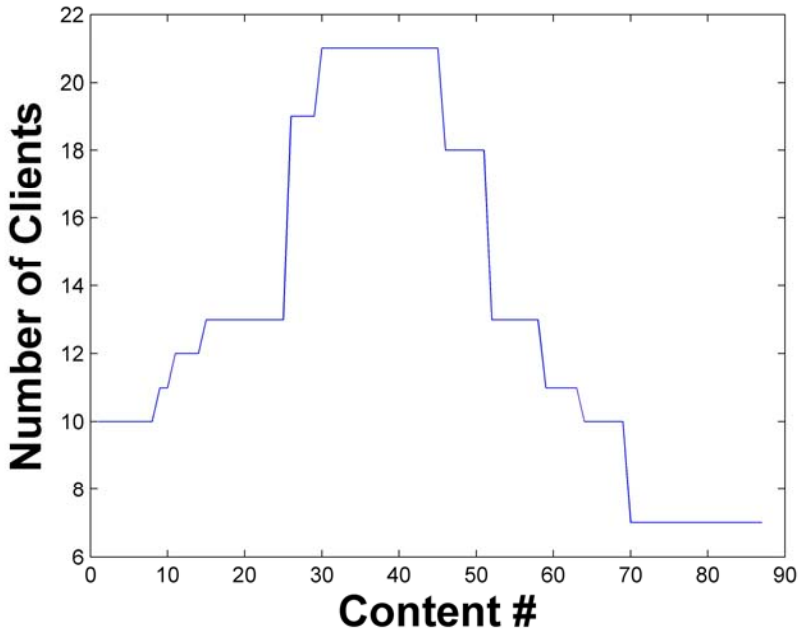

Figure 5.2 TuVista number of clients versus time.

Since clients receive new content notifications by polling the server, the latency between when new content is available on the server and when clients receive it is dependent on their polling interval. The total mean latency for all clients over all content notifications was 0.71 seconds (Figure 5.3). Latencies tended to increase as more clients used the system due to increased server load and increased response time.

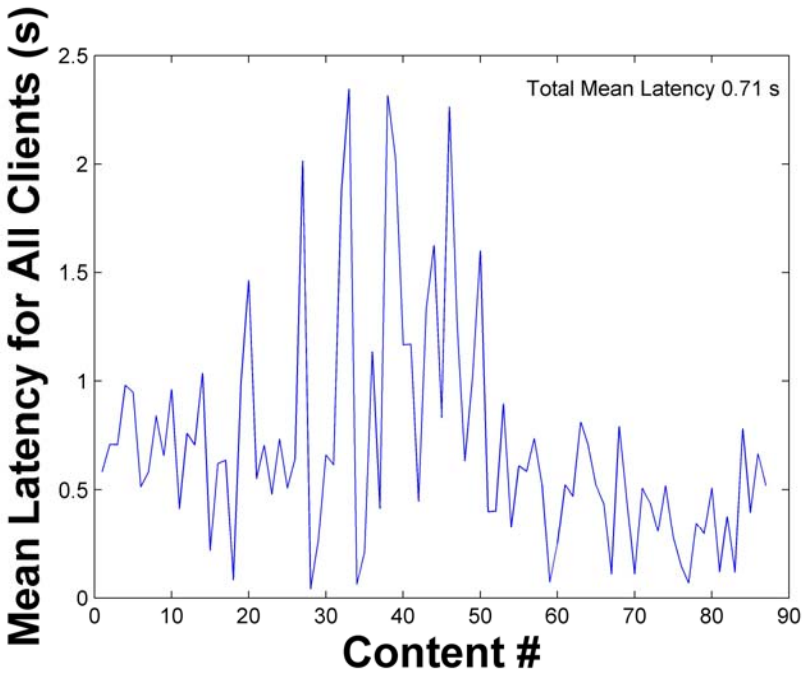

Figure 5.3 TuVista mean client latency versus content number.

5.2 MobiSportsLive

MobiSportsLive is a bi-directional content aggregation, creation, and push distribution platform for mobile devices. Its content aggregation and distribution foundation is the NRTPM. Its capabilities include automated content generation, user generated content, collaborative content sharing, and device specific push content distribution. Mobile clients act as content consumers as well as producers.

5.2.1 Literature Review

The mobile application's architecture is based on existing work [35, 42, 43]. The architecture and implementation of the middleware and overall system also draw from previous research [36, 37, 38, 40, 41].

5.2.1.1 Mobile Internet for the Multimedia Enterprise

An architecture for multimedia collaboration for mobile devices was developed in the Mobile Internet for the Multimedia Enterprise (MobIME) project [35]. A three layered architecture is proposed: 1) Content Representation, Delivery, and Management 2) User-Device Interaction 3) Device Resource and Bandwidth Management (Figure 5.4). The Content Representation, Delivery, and Management layer is subdivided into its three major components. Voice, 2D/3G graphics, and text content are represented by the µ3dML format which is based on SVG [34, 41].

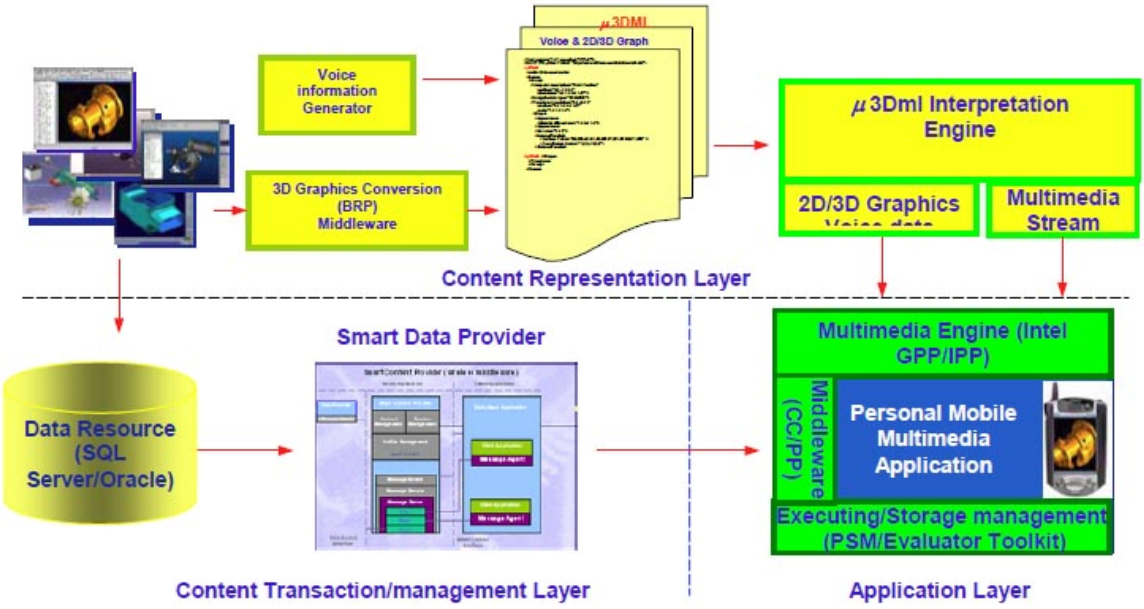

Figure 5.4 MobiME system architecture.

Multimedia content is Base64 encoded and encapsulated in the µ3dML container. Content Delivery occurs over an HTTP connection. Content management is handled by a Smart Data Provider that stores and retrieves content references from a SQL/Oracle database. User-device interaction allows users to view, notate, and share content with other users – in this way collaboration is facilitated. The Device Resource and Bandwidth Management layer handles

reformatting of content for the requesting device. When a client requests content, this layer creates a custom version of the content and encapsulates it in μ3dML format. The client application is able to process content using its μ3dML processing engine that parses the XML container and displays the content and associated meta-data. The architecture and content and device management concepts from this work were used in the MobiSportsLive project.

5.2.1.2 In the Direction of a Sensor Mapping Platform Based on Cellular Phone Networks

Location-based services are described in the work "In the Direction of a Sensor Mapping Platform Based on Cellular Phone Networks" [42]. Location capabilities are built-in to all mobile devices since 1999 as mandated by the FCC's Wireless Communications and Public Safety Act [44]. This is accomplished by either cellular signal multilateration or GPS positioning. When GPS signals aren't available, e.g. when the device is indoors, cell tower multilateration can yield a location with an error ranging between 10s of meter to 1000s of meters [46]. In contrast, GPS can achieve accuracy of up to 2 meters, although GPS positioning uses more energy due to the relatively high power requirements of GPS chips [45]. Current Smartphone platforms including iPhone and Android provide APIs that seamlessly determine device location using both methods depending on availability [47, 48].

5.2.1.3 Middleware for Multimedia Mobile Collaborative System

Middleware for Multimedia Mobile Collaborative System [37] presents a specialized middleware for mobile devices enabling collaboration, with a focus on content generation, delivery, and representation. The framework consists of three layers: 1) Content Generation 2) Communication 3) Content Consumption/Regeneration. In the Content Generation layer the server generates an

appropriately formatted version of the content for the requesting device. The device request consists of the device profile (user agent), previous network status (connection bandwidth), and requested URL. The Communication Layer handles client message delivery and maintains state information including device profile, connection status, and message delivery information. The Content Consumption layer exists on the mobile application and allows clients to view, annotate, and modify the content. The Content Regeneration layer, which is server-side, receives client modifications, regenerates the content, and automatically updates other clients – in this way real-time collaboration is facilitated.

5.2.1.4 A Service Oriented Middleware for Seamless System Nomadic-Aware Servants

Service Oriented Middleware for Seamless System Nomadic-Aware Servants [36] describes a client-based middleware that seamlessly discovers and provides location dependent information services (LDIS) to users. LDIS may have non-standard interfaces or be deployed dynamically. The middleware automatically handles roaming among services and discovery. Client requests are seamlessly adapted to services' interfaces – in this way the client software can be written generically and service interfacing is handled by the middleware.

5.2.1.5 DMW – A Middleware for Digital Rights Management in Peer-to-Peer Networks

A middleware for enforcing DRM policies in a collaborative system is proposed by the DMW research project [38, 40]. Content sharing among users in a collaborative system must be uninhibited enough to ensure a quality user experience, but provide enough limitations such that content providers can still capture their fair share of revenue. The proposed middleware would exist on client devices and has a three tiered architecture: 1) Application 2) OS 3) Hardware.

Each tier is comprised of multiple service-oriented layers (Figure DMW Arch). The Application tier is includes the User, DRM, and Directory Services Layers. The User Services Layer handles user authentication, content information exchange and compatibility analysis. Content information exchange entails sending and receiving of content meta-data including content ID, format, device requirements, and DRM specifications. Compatibility analysis ensures that devices only receive content that can be rendered on their device and that can enforce DRM. The DRM services layer handles content acquisition, rights analysis, access control, and copy control. Content acquisition downloads content after user authentication and content compatibility have been performed. Access control determines how many times and for how long the content may be rendered. Copy control limits the number of duplicates can be created. The Directory Services layer maintains the list of available content and delegates content authorization. The Device Services layer ensures the device has the necessary resources (RAM, processor speed) to render a piece of content, and enforces the DRM policy provided by the DRM Services layer. The Transmission Services layer controls the transmission of content over different networks e.g. if the user is connected to a Bluetooth and WiFi network, a content item's DRM policy may dictate that he may transmit it over Bluetooth but not WiFi.

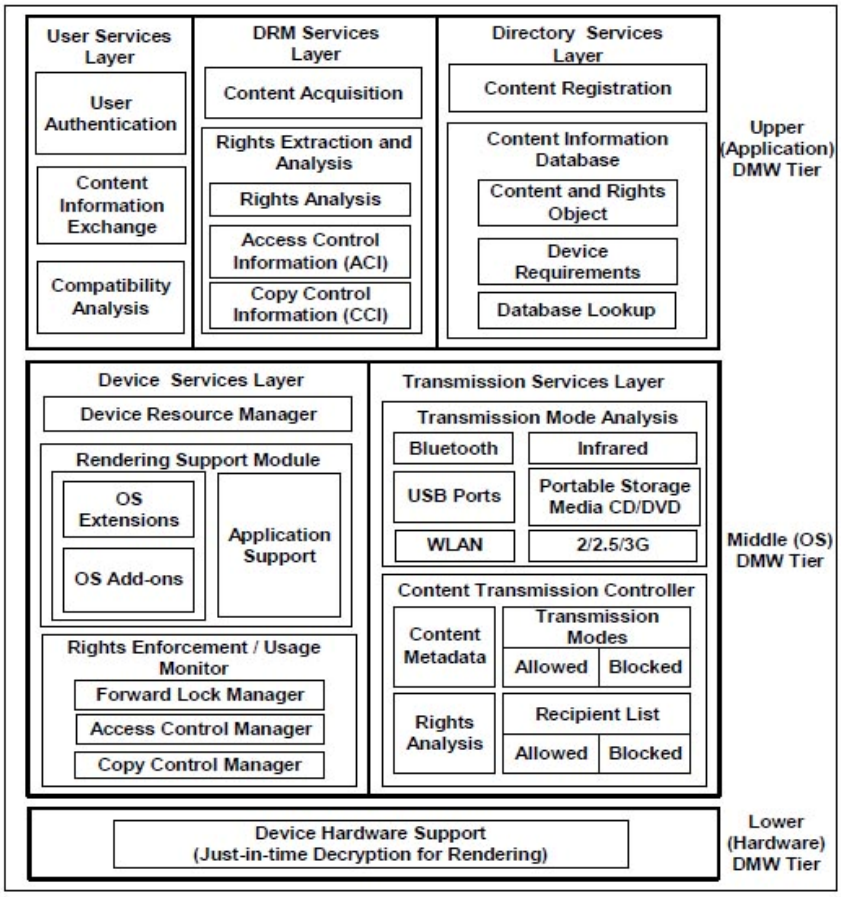

Figure 5.5 DMW layer architecture.

5.2.2 System Architecture

MobiSportsLive is comprised of multiple discrete components that handle specific tasks (Figure 5.6). A live video feed is encoded using the H.264 video codec and MP3 audio codec, contained in an ASF file, and streamed over RTP to the Automated Content Creator (ACC). Created content along with extracted meta-data is sent to the NRTPM. The NRTPM handles content transcoding, indexing, addition to the database, and push distribution of incoming content.

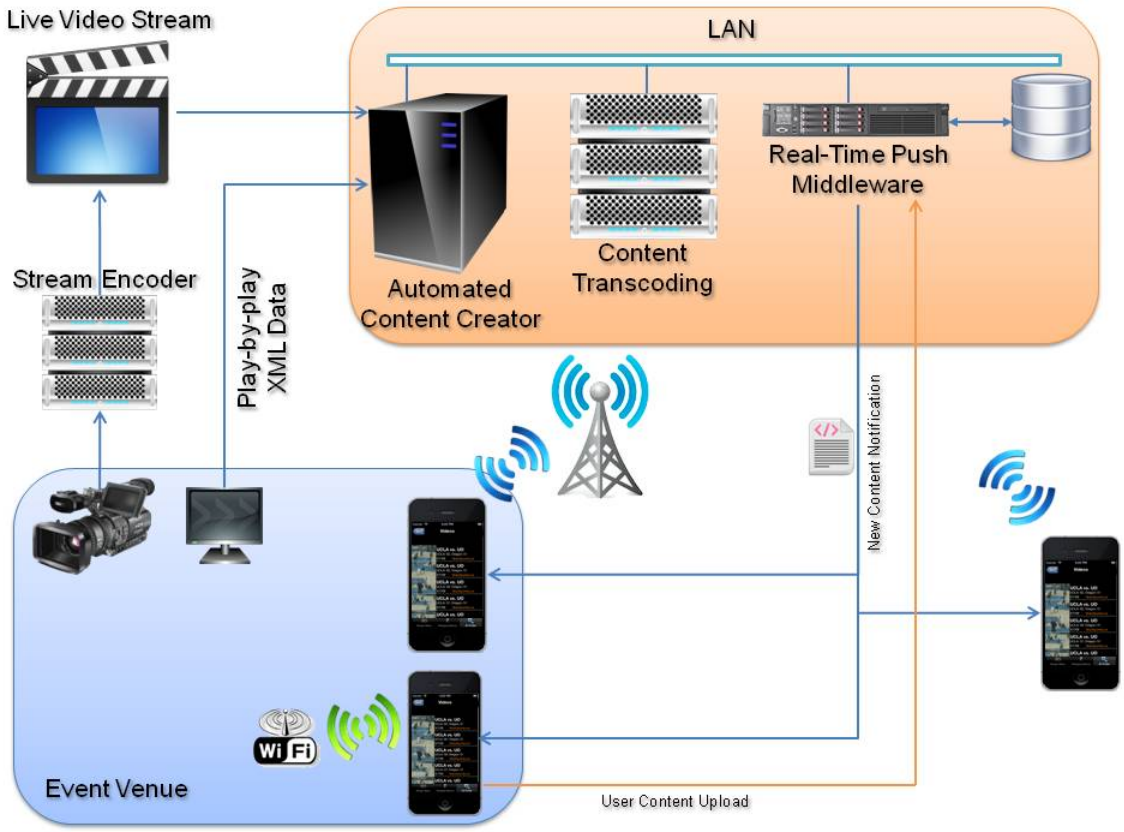

Figure 5.6 MobiSportsLive system architecture

5.2.2.1 Components

5.2.2.1.1 Automated Content Creator

Using a real-time video stream and real-time XML feed of events, the MobiSportsLive Automated Content Creator (ACC) creates video clips of the pertinent action and associates descriptive tags. Based on configurable keywords, the ACC will create a video clip when a match is found in the real-time XML event feed. The arrival time of the XML event update is taken as the clip's end time, while the start time must be extracted from the event and synchronized with the real-time video stream's timestamp. Using the FFMpeg command line video manipulation tool, the ACC extracts the appropriate clip then sends it along with meta-data

including descriptive tags, and location, to the NRTPM for indexing, addition to the database, and distribution to clients.

5.2.2.1.2 NRTPM

The NRTPM is the heart of MobiSportsLive's content aggregation and distribution engine. It indexes and stores incoming content from the dedicated providers, the ACC, and mobile clients. Content is pushed to mobile clients based on their location and tag criteria.

5.2.2.1.3 Mobile Application

A mobile application was developed for user content generation and consumption. Clients set their preferences in the form of a subscription detailing the description and originating location of content they wish to receive. In addition to dedicated content providers, clients can create and upload content with descriptive tags and location information.

5.2.2.1.4 Architecture

The MobiSportsLive mobile application is built on a 4 layer architecture which includes the UI, Application, Data, and Connectivity layers (Figure 5.7)

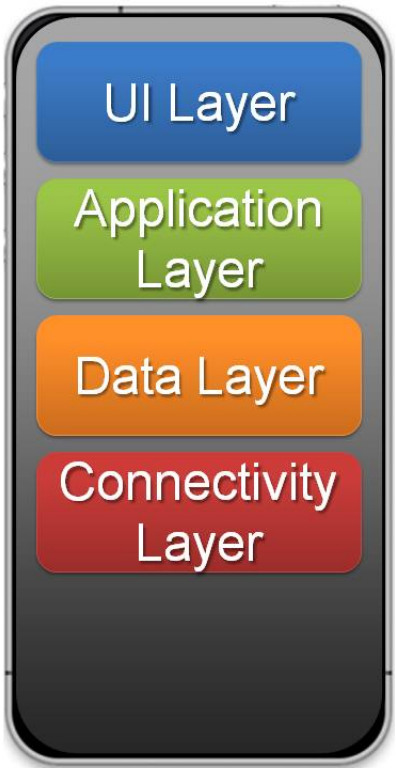

Figure 5.7 MobiSportsLive mobile architecture.

5.2.2.1.4.1 UI Layer

The UI Layer handles presentation to and input of data from the user. The three primary views presented to the user are 1) Subsciption Settings 2) the Live Content Portal 3) the Content Upload portal (Figure 5.8). Subscription settings are retrieved and parsed by the Application Layer from the Data Layer and given to the UI Layer for display and editing. New content is fed to the UI Layer from the Application Layer and presented as rows in a table. A thumbnail of the content is displayed, along with meta-data including the date, time, content owner, and event data including event name, team names and score. The Content Portal allows the user to take a photo or video or select an existing one from their library and upload it. They can also choose to include the location of the media and add descriptive tags to it.

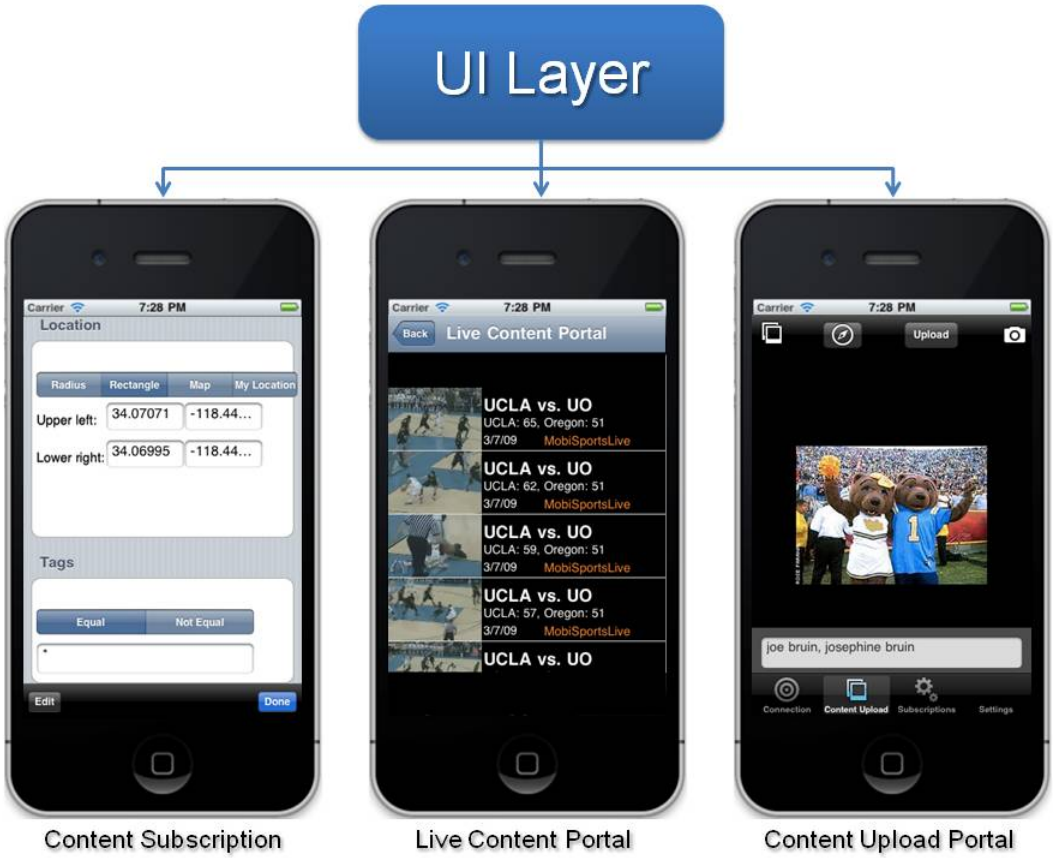

Figure 5.8 UI Layer.

5.2.2.1.4.2 Application Layer

The Application Layer handles high level delegation of tasks to the lower layers and logic functions that directly support the UI Layer (Figure 5.9). User subscriptions are parsed by the data layer and fed to the UI Layer by the Application Layer. When changes are made by the user, the Application Layer then sends the data to the Data Layer so an XML representation can be generated for transmission to the NRTPM. Content capture is handled by invoking the media

API which allows the user to capture new or choose an existing photo or video. The content is then sent to the Data Layer for encoding and to the connectivity layer for upload.

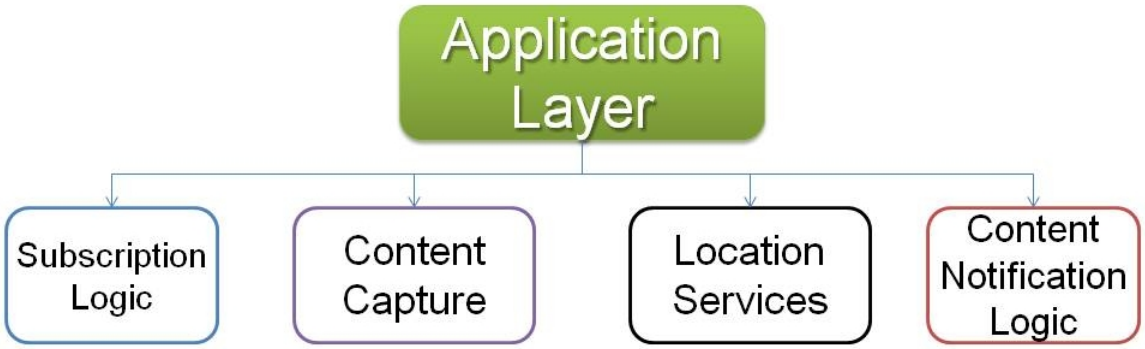

Figure 5.9 Application Layer.

5.2.2.1.4.3 Data Layer

The Data Layer handles data storage, retrieval, and transformation (Figure 5.10). Incoming socket messages are processed and sent to the Application Layer for dispatch. Outgoing messages are assembled and sent to the NRTPM. User subscriptions and profiles are stored locally on the device in XML format. When they are requested by the Application Layer, the Data Layer parses and returns the stored XML. After changes have been made, the Data Layer stores the updated files in XML format. Content is Base64 encoded for upload over HTTP.

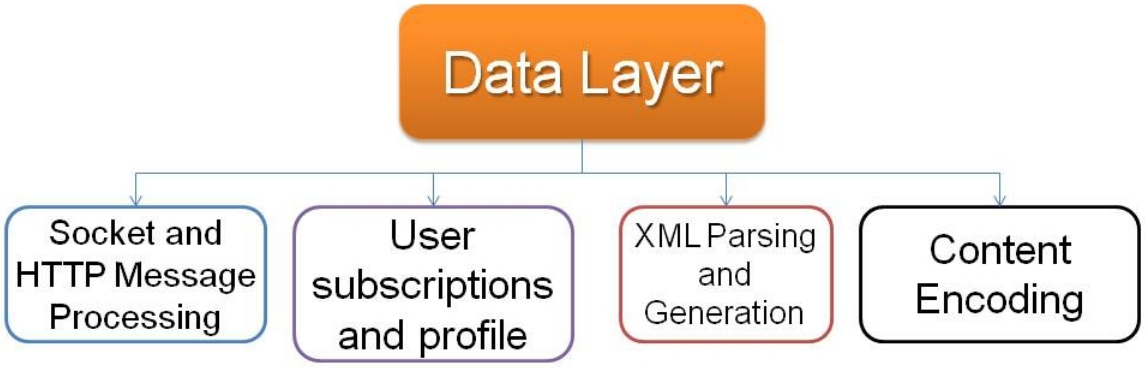

Figure 5.10 Data Layer.

5.2.2.1.4.4 Connectivity Layer

The Connectivity Layer handles socket connections and disconnections to the NRTPM, sending of outgoing messages, and passing incoming messages to the Data Layer for processing (Figure 5.11). Content uploads are sent over HTTP.

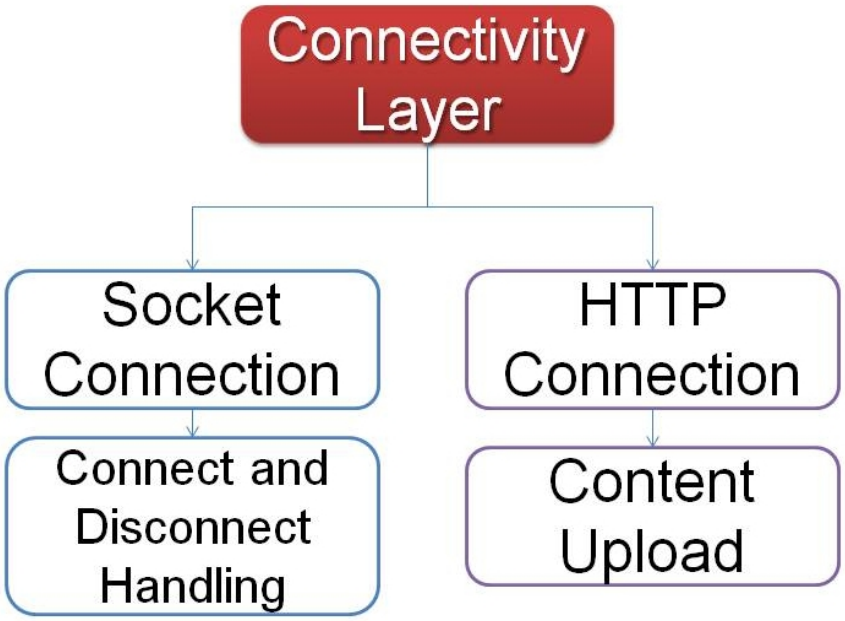

Figure 5.11 Connectivity Layer.

5.2.3 System Trial

5.2.3.1 Trial Setup

MobiSportsLive went live in a real-world trial during a UCLA basketball game at Pauley Pavilion. A control group of 30 students and researchers used a combination of iPhones and iPod Touches to test system performance. A total of 10 users connected via 3G, 15 via in-venue WiFi, and 5 via WiFi from remote locations outside the venue. In-venue WiFi users were spatially distributed so no more than four users were connected to one access point. The ACC and Content Transcoding were performed on a dual Xeon 3.0 GHz with 1 GB RAM located in the WINMEC lab. The ACC was fed a live video stream and live play-by-play event stream. Video was transcoded using the H.264 codec, 30 frames per second, 320x240 resolution, AAC audio, mp4 file format, in two bitrates: 100 kbps and 300 kbps.

5.2.3.2 Results

Clip creation and transcoding were combined into a single FFmpeg call and yielded a median value of 42 seconds to process 30 second clips with 100 kbps and 300 kbps bitrates. The content store was on the same LAN as the ACC so transfer time was negligible. Time taken by the NRTPM to add and index content averaged 25.15 ms over the 40 content items added (Figure 5.12).

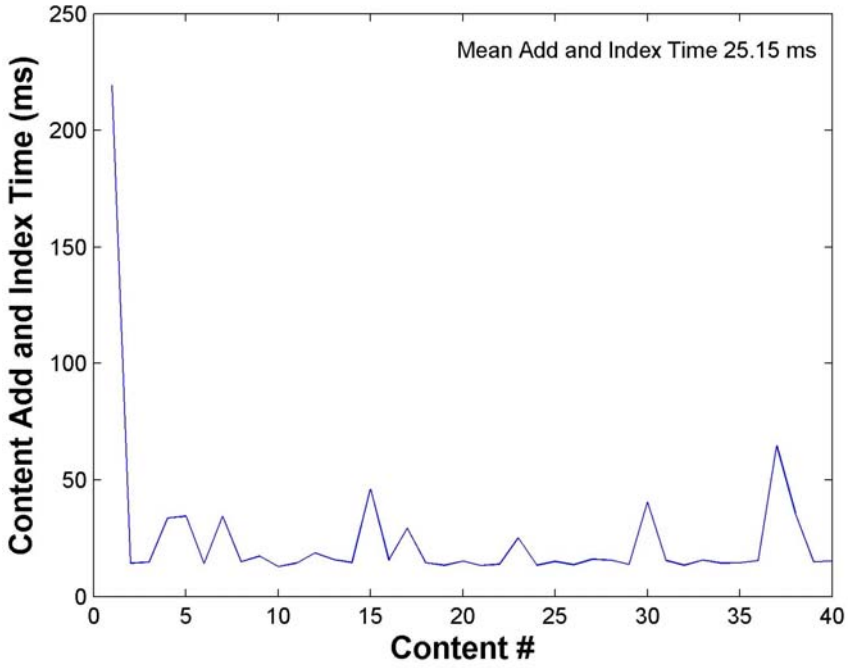

Figure 5.12 NRTPM content processing time.

The total time to create a clip inclusive of ACC processing time, transcoding time, transfer time, and server application time, was around 45 seconds. After each content item was added and indexed by the NRTPM, client subscriptions were searched for a match. Each client was provided a default subscription comprised of one Rule with only a location condition bounded by the upper right coordinate (34.068333, -118.444609) and lower right (34.067778, -118.443375). The latency was measured between the time content adding and indexing completed and when an acknowledgement message was received from each client. Clients were segregated by connection type and location and the mean latency was calculated for each content item. Fifteen In-Venue WiFi clients had a mean server ping time of 15.0 milliseconds and a mean content notification latency of 49.5 milliseconds over all 40 content item notifications (Figure 5.13). Five Outside-Venue WiFi clients had a mean server ping time of 29.0 milliseconds and a mean content notification latency of 89.7 milliseconds (Figure 5.14). Ten 3G connected clients, who

were located both in and outside the venue, had a mean server ping time of 103.0 milliseconds and a mean content notification latency of 249.9 milliseconds (Figure 5.15). Latency results for each content item are averaged for all clients, and the total mean latency represents the mean over all content items for all clients. Each latency time is the sum of the time taken for the content notification message to travel from the server to the client, and the time for an acknowledgement message to travel from the client to the server – the true message notification latency would be less by approximately the ping time from the client to the server.

Compared to the performance of the TuVista system, MobiSportsLive created and transcoded clips automatically in 35% less time (45 seconds versus 70 seconds). This was due to the ACC's efficient method of clip creation and transcoding and the system architecture's network topology. Latencies for in-venue clients were lower by 93% (49.5 milliseconds versus 710 milliseconds). The greatly reduced latency is attributed to the NRTPM's push notification delivery. MobiSportsLive proved superior as an end-to-end system for near real-time content delivery to mobile devices. Its automated instant replay creation obviates the need for a human to manually create clips on a dedicated workstation. Its push content delivery via the NRTPM reduces content notification latency to an average of 49.5 milliseconds and eliminates the need for mobile clients to waste battery power by continuously polling a server for new content.

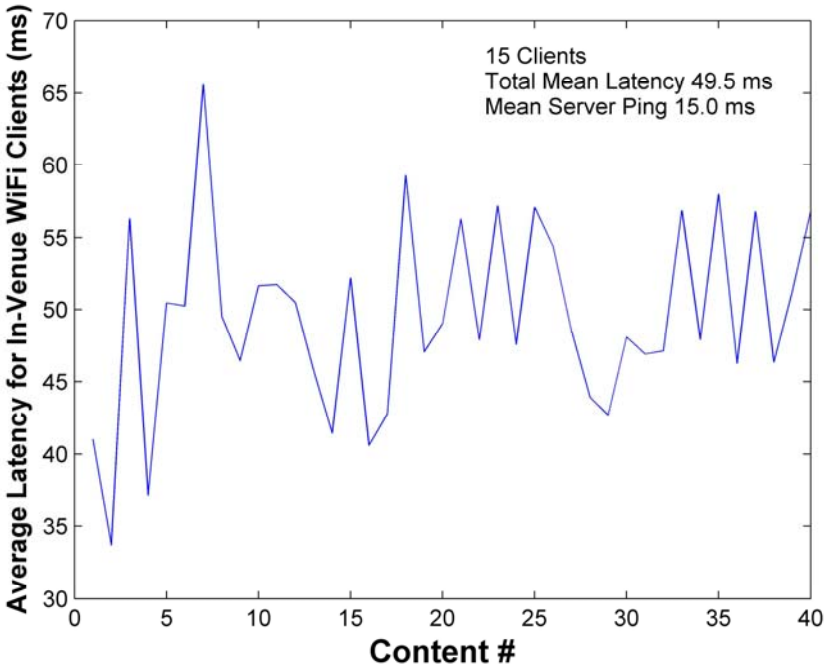

Figure 5.13 Content notification latency - in-venue WiFi clients.

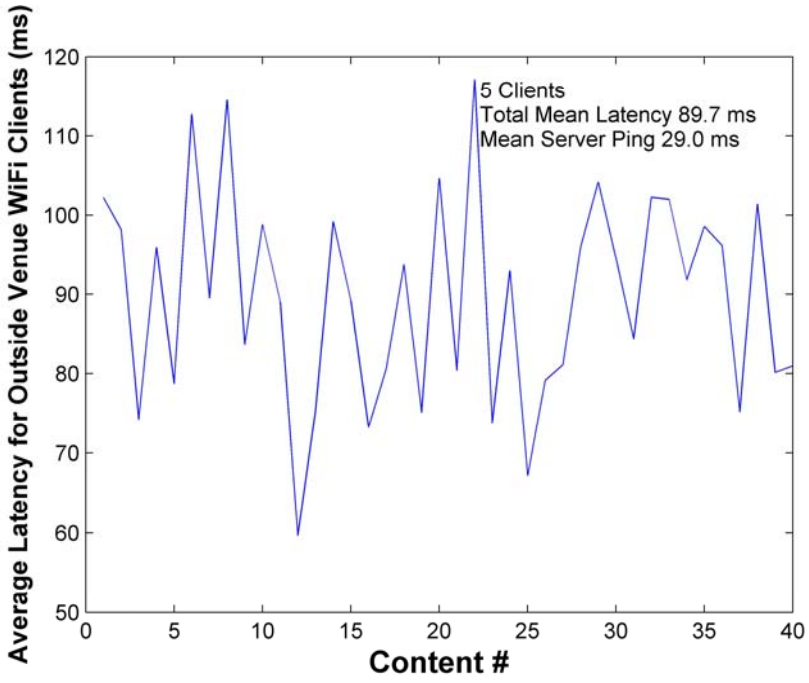

Figure 5.14 Content notification latency - outside venue WiFi clients.

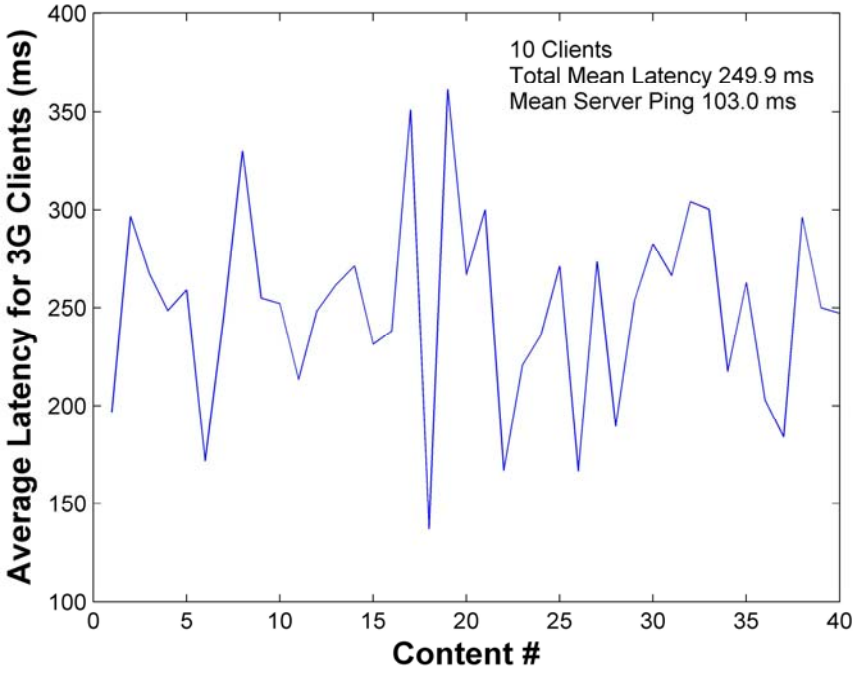

Figure 5.15 Content notification latency - 3G clients.

Chapter 6 Electric vehicle smart charging and vehicle-to-grid operation

One million electric vehicles (EVs) and plug-in hybrid electric vehicles (PHEVs) are expected to be in use by individuals and fleets by 2015 [53]. Unmanaged EV charging will add to peak grid load and would require additional generation capacity [54, 55]. Charging must be scheduled intelligently in order to avoid overloading the grid at peak hours and to take advantage of off-peak charging benefits. EVs can also serve as an energy resource through vehicle-to-grid (V2G) operation by sending electricity back into the grid thereby preventing or postponing load shedding [56, 57]. Charging and V2G services must be optimized for grid load while guaranteeing owner schedule and range requirements are met.

A system leveraging mobile devices and application to facilitate "smart" charging has been proposed [58], however integration of the mobile component with a charge scheduling component is not specified. In addition, the described system does not account for discrepancies between specified user charge profiles and actual distance traveled and times of arrival and departure. A conceptual framework for V2G implementation has been developed [57], however EV owner input into the system has not been considered. The market penetration of smart technologies and Advanced Metering Infrastructure (AMI) has resulted in smarter EV charging techniques which minimize charging cost to the consumer and grid load at peak hours. Shao et al. [59] proposed a quick charging method where a higher load (240V, 30A) is drawn to enable quicker charging during evening (after 6 p.m.) and off-peak hours. Home-based and off-peak charging (9pm to 11 am) is also considered by Yu [60].

6.1 Literature Review

6.1.1 Radio-Frequency Identification (RFID)

RFID has been playing an important role in various kinds of supply chain based industrial applications such as warehousing, pharmaceutical tracking and retailing since the past decade. An RFID system consists of one to many readers which communicate with multiple tags by inquiring their identification (ID) [61]. This technology has been widely adopted in supply chain systems to streamline the flow of information regarding identification, arrival/departure and status of objects in the system. Tags are periodically queried by their respective readers, which in turn notify a software service/middleware of their presence or absence, which can be further processed to deliver useful service. RFID tags are separated into three categories based on their energy usage: Passive, Active, and Semi-Passive depending upon their power consumptions [62]. RFID tags are also categorized based on the frequency at which they operate such as Low Frequency (LF), High Frequency (HF), Ultra High Frequency (UHF) and Microwave depending on the application [63].

RFID technology has been used for some time in parking lots in the form of Near Field Communication (NFC). The typical role of NFC in parking lots is to grant entry to authorized users. Our system would employ an RFID reader at every access gate to read an entering vehicle's tag. Once the tag's ID has been read, it is transmitted to the system middleware which performs a database lookup. The middleware will either grant the vehicle access and assign it to a parking spot or deny access. Porter *et al.* discuss the implementation of RFID based vehicle mileage logger at gas stations [64]. They discussed rates of getting successful reads by the RFID mileage loggers from devices placed in vehicles and the issues that affect the read rates of the

RFID readers. They also discussed an architecture where a middleware bills the drivers of the vehicles with the toll based on the distance they drove on toll highways after collecting the data through the RFID readers. They also implemented GPS based technologies to supplement the mileage collection process. Theo *et al.* discuss the concept of an Energy Name Service (ENS) for the Smart Grid energy infrastructure [65]. The concept they have explained is inspired by the principle of Domain Name System (DNS) and Object Name System (ONS) which is very closely related with the concept of internet of things using RFID tagged objects. The philosophy of their work is that how every entity in the energy domain (like charging stations, vehicles etc) would be given an ID, which would be helpful in identification and hence execution of seamless transactions for charge consumption between multiple geographical entities. This would be similar to the mobile phone service where a user would be billed for their usage regardless of the location of usage of the mobile phone service. They also introduced the concept of Internet of Energy similar to the internet of things where every object related to the energy food chain is interconnected to each other through the internet. Their paper discussed the architecture of such an energy network and methodology for the energy naming service where an object is given an energy ID based on certain parameters such as geographical location, type of object etc. Song discusses the implementation of a home electricity management box installed at every household which acts as an intelligent node to optimize the consumption of electric energy [66]. They propose using the RFID enabled car keys as instruments to authenticate EVs for charging.

6.1.2 Charge Scheduling

Soares *et al.* have proposed a Particle swarm optimization (PSO) based approach to perform V2G based charge scheduling [67]. Their charging plan also encompasses distributed power

generation systems such as fuel cell, solar etc which contribute to the net power generation. Their goal is to minimize the generation cost which includes generation production cost and V2G discharge payment. They compare their optimization approach with General Algebraic Modeling System (GAMS) based optimization software. However, they don't explain the technique employed by the GAMS software to optimize the charge scheduling which makes it challenging to assess the two techniques. In the final result they show that the GAMS based software results in lower total costs but the PSO based approach has a lower execution time. However, this claim is also hard to validate as very little light has been thrown on the technique employed by the GAMS optimization software. Venayagamoorthy *et al.* proposed a real-time digital simulator based vehicle parking lot performing V2G transactions [68]. They have used a binary particle swarm optimization technique to control the power flow to and from a vehicle. The goal of their optimization technique is also to minimize the cost of charging a given EV. In this paper, they analyze the effects of large bidirectional power surges to the batteries and the inverters of individual plug-in vehicles as they are switched from a state of charging to discharging. They concluded that grid faults can be detrimental to vehicles performing V2G transactions unless advanced control and protection is provided. Hutson *et al.* extended the work by Venayagamoorthy on V2G based charge scheduling in parking lots [69]. The charge optimization was again based on the BPSO algorithm previously discussed. They considered two cases. The first case study takes the price curve and finds the best selling price for each vehicle over the desired departure State of Charge (SOC) and the best buying price for each vehicle under the desired departure SoC. Case 2 allows for multiple transactions to occur for each vehicle throughout the day leading to more profit oriented charging. However, the Binary PSO (BPSO) algorithm being stochastic, the same solution is not found each time leading to a

standard deviation of 0.045% of the average. Wu *et al.* introduced the concept of intragrid which aggregate the contributing EVs under an umbrella to act as a single unit feeding and taking energy from the grid [70]. They considered different scenarios where either the cost of charging the EVs or the emission from the charging or both can be minimized. The optimization algorithm used is Particle swarm optimization. They achieved their optimum solutions after 1000 iterations. Saber *et al.* propose the concept of an intelligent Unit commitment (UC) by using V2G to reduce operational costs and emissions [71]. BPSO was used to perform scheduling of thermal units while Balanced PSO was used to determine the number of V2G units to be connected to grid to minimize the cost of operation and emission by the thermal units. Different scenarios were simulated by optimizing the cost, emission and combined goals. Schieffer *et al.* have proposed a decentralized charging strategy in their work [72]. First they employ linear programming to optimize charging duration by minimizing charging events during peak load (or high priced) hours. On completion of this step they use probability density functions indicating the distribution of charging slots to determine the exact time choices for charging. They have many similarities with our work such as charge time optimization, division of entire day into smaller charging/discharging time intervals etc which would be highlighted in greater detail during algorithm discussion.

6.1.3 Vehicle-to-Grid (V2G)

Guille and Gross [57] present a conceptual framework for implementation of V2G based on bi-directional energy transfer between vehicle and grid and aggregated use of EVs as generation and storage devices. Aggregated EVs can provide grid services such as up and down regulation, load leveling, and peak shaving more economically and with less environmental impact than

current systems. EVs must be aggregated because individually their battery capacity is small and would not make an appreciable difference at the grid level. In contrast with unmanaged EV charging that can add to reserve and regulation requirements, aggregated EVs can be charged during off-peak periods thereby levelizing grid load and reducing these requirements. The proposed framework emphasizes the Aggregator as the enabling entity for V2G. The Aggregator has the following communications relationships: 1) each EV sends status data and receives charging control signals 2) the Energy Service Provider (ESP) receives charging and sends load levelization requirements 3) the independent system operator or regional transmission organization ISO/RTO sends resource requirements and receives resource availability data. This work presents seminal ideas for future V2G work, however specific components of the proposed framework are neglected including EV owner input into the system and charge scheduling for random arrival and departure times.

Kempton and Tomic [56] present profit calculations for using EV fleets for V2G services including up and down regulation. Assuming an availability of 17 hours per day for 252 RAV4 EVs, an initial state-of-charge of 30-50%, a required range of 36 miles, and circuits able to handle 6.6 kW, profits range from $144,800-912,000 for regulation down/up market prices from 12.9/14.0-39.7/62.5 US$/MW-h. When a 15 kW limit is considered the range is $358,000-2,102,000. The results include equipment costs and battery life degradation.

Han et al. [73] present a method for optimally controlling EV charging to maximize EV regulation service revenue. The model developed accounts for varying electricity and regulation price over time, a variable time the vehicle will be parked, a variable amount of charge needed, and a maximum charge rate. A maximum revenue of $.42 is determined assuming a 20 kWh

battery, maximum charge/discharge rate of 2 kW, an arrival time of 00:00, and departure time of 12:00.

6.2 System Architecture

EV owners will register one or more vehicles with the system. Each registered vehicle will wear an RFID access tag that will allow it to enter enterprise owned and affiliated garages. The tag's unique ID is used by the Parking Garage Aggregation Middleware (PGAM) to lookup the associated owner and act on his/her charging profile and billing information. At the access gate (Figure 6.1) the driver will be designated a numbered parking spot based on availability. Once the owner plugs in to the EV Supply Equipment (EVSE), the PGAM checks for an existing charging profile. If none is found he/she will be prompted to enter one via the EV Command Portal (EVCP) application on their mobile device or charge station touchscreen (Figure 6.2).

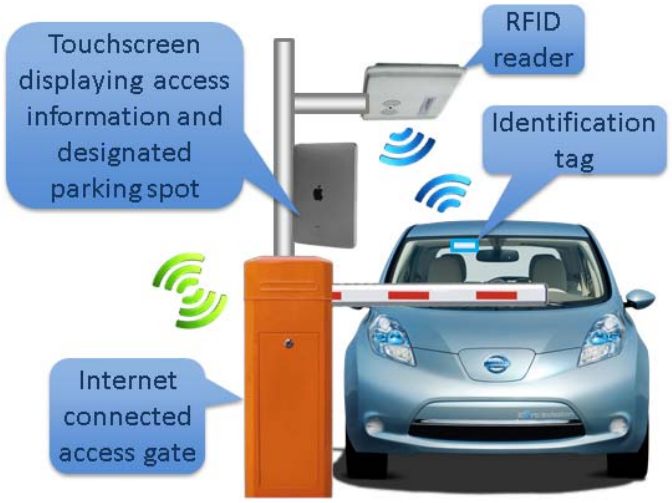

Figure 6.1 Parking garage access gate.

Figure 6.2 Charging station equipment and activities.

Figure 6.3 shows the sequence of events when a car arrives at the gate. If the user fails to enter a profile within a given time window, the system uses a default. The EV owner can use the EVCP to monitor the status and control the charging modality of his/her vehicle within system parameters at any time. Based on user charge profile the Aggregated Charge Scheduler (ACS) will schedule charging to meet user charge requirements, minimize cost, and maximize profit from V2G services. Figure 6.4 shows the flow of the aforementioned data among system components.

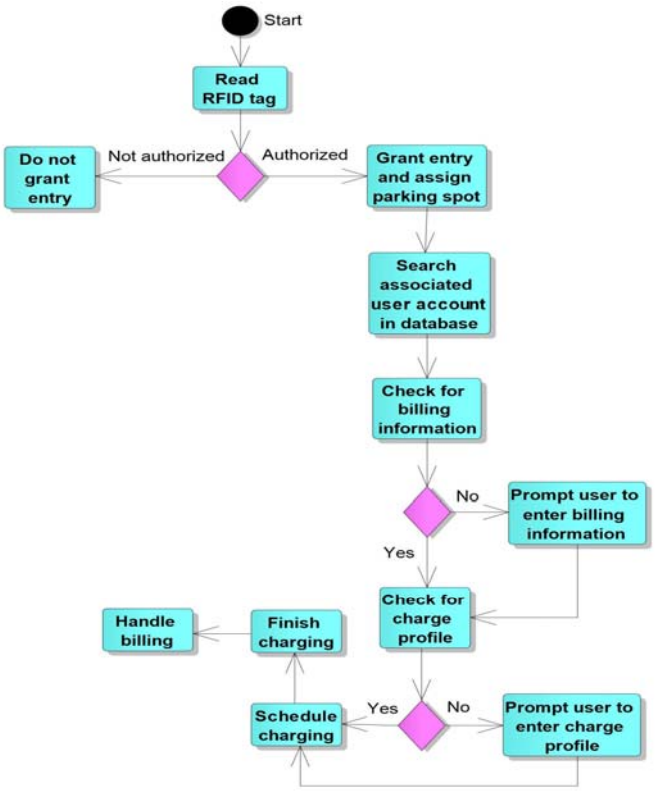

Figure 6.3 Event sequence for EV arrival at parking garage.

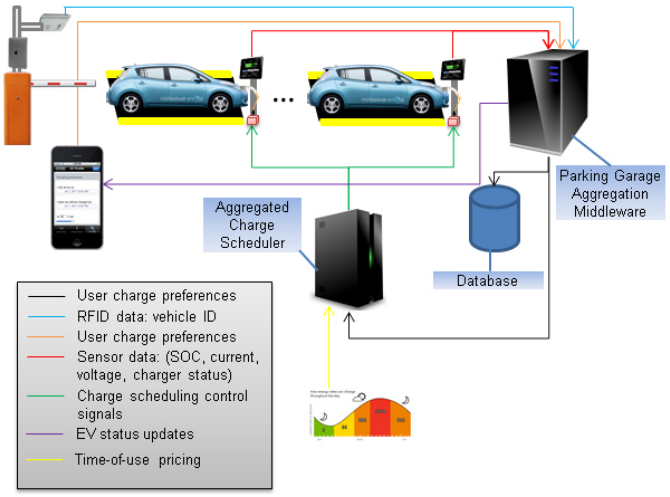

Figure 6.4 Data flow among system components.

6.2.1 EV Command Portal Application

Range anxiety is one of the main obstacles to consumer adoption of EVs. The problem stems from 1) limited range compared to conventional gasoline vehicles 2) inadequate charging infrastructure [74]. This anxiety can be mitigated if EV owners have better access to and control of the charging of their vehicles. This control can be achieved intuitively via a web or mobile application. Users should be able to monitor their vehicle's state of charge (SOC), range, estimated charge completion time, and estimated cost of charging. They should be able to control the charging of their vehicle using parameters including desired SOC, time of arrival, time of departure, and, if available, V2G opt-in.

The EV Command Portal (EVCP) application will play an integral role in the scheduled charging of EVs by allowing owners to monitor the status of their vehicles as well as control the way they charge. The application will provide the user with real-time updates on the SOC of their battery, real-time charge status alerts e.g. charge completed, and allow users to control the charging modality of their vehicle by creating charging preferences and schedules.

The EVCP interface is comprised of three screens: 1) Charge Status 2) Charging Stations 3) EV Profile. "Charge Status" is the main screen and displays the EV's SOC, range, time remaining until charging is completed, current electricity price, and estimated total charging cost (Figure 6.5). The "Charging Stations" screen displays charging stations on a map along with charger type and real-time availability information. The "Charge Profile" screen displays a calendar and a list of existing profiles where the user may create one or more profiles per day. The profile creation screen allows the user to enter arrival and departure times, initial and final SOC values, and their V2G participation preference (Figure 6.6).

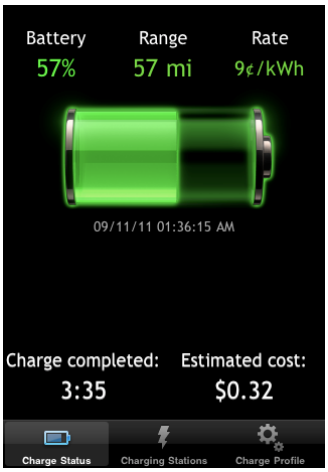

Figure 6.5 Charge status screen.

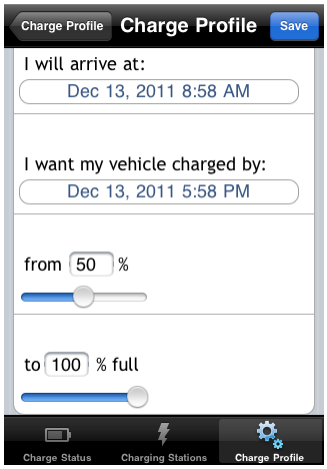

Figure 6.6 Charge profile screen.

6.2.2 Parking Garage Aggregation Middleware

The Parking Garage Aggregation Middleware (PGAM) is a lightweight middleware that handles parsing of user charging profiles, aggregation and dissemination of charger and vehicle status, billing, and alert notification delivery [82]. A vehicle's RFID tag is scanned at the access gate and the ID is sent to the PGAM which looks it up in the system database. If it is authorized the PGAM assigns a parking spot, shown on the gate's display screen, and signals the gate to open.

The middleware then checks for billing information and a charge profile. If either is missing, the owner is prompted to enter the required information on his/her mobile device or via the touchscreen at the charging station. The EVSE sends charging voltage and current data to the PGAM. The vehicle's SOC is estimated using the ISOC provided by the user, charging power as a function of time, and the vehicle's battery charge profile. Charging cost is calculated using power draw/supply data from the EVSE, electricity price as a function of time, and the vehicle's charging schedule. EV updates are pushed to the EVCP over a TCP socket connection (Figure 6.7). A buffer time, *T_buff*, is provided between when the ACS schedules cars to be charged to the desired SOC and the specified departure time. By default this value is equal to zero. Based on the difference between specified departure time and actual departure time, provided by the RFID reader at the garage exit, the middleware will alter the value of *T_buff* for a given driver. A charge profile is received from the EVCP in XML format, parsed by the middleware, and sent to the ACS. If the profile is verified by the ACS the PGAM adds it to the database. If the ACS determines the charge profile cannot be satisfied it sends an error message to the PGAM, which sends an alert to the user, detailing the parameter(s) and corresponding value(s) to change.

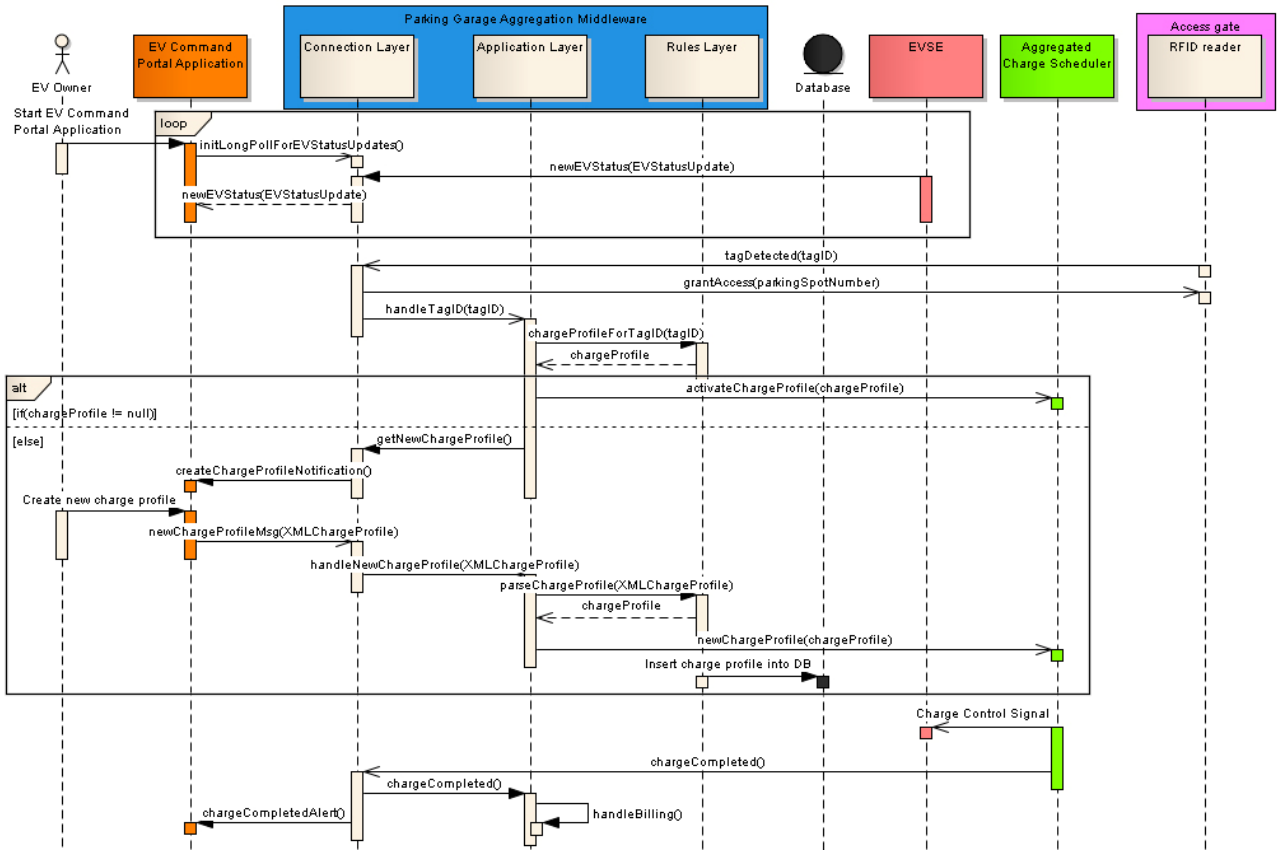

Figure 6.7 System sequence diagram.

6.3 Aggregated Charge Scheduler

The Aggregated Charge Scheduler (ACS) optimizes EV charge scheduling for minimal cost using user charge profiles and electricity price as a function of time. By optimizing charge scheduling for electricity price it is implicitly optimized for electricity demand. The ACS sends a control signal to each EVSE to charge, discharge, or turn off according to the created schedule. The ACS receives an owner charge profile from the PGAM which includes time of arrival (*t_arr*), time of departure (*t_dep*), buffer time, *T_buff*, initial state of charge (*ISOC*), and final state of charge (*FSOC*) of their EV. Charging must be completed *T_buff* minutes before the

owner's scheduled departure. The value of *T_buff* is calculated by the PGAM based on owner adherence to his/her specified departure schedule. Once the ACS receives a charge profile its task is to charge the EV from *ISOC* to *FSOC* within the time span of *t_arr* and *t_dep – T_buff*.

6.3.1 Algorithm Description

6.3.1.1 Charge Scheduling

The 24 hour period of a day is quantized into smaller time intervals which are the smallest units of time used by the algorithm to schedule the occurrence of a charge, discharge or no-activity. The electricity price curve (Figure 6.9) is also quantized into time intervals where each interval has a fixed price, P_i. Using the charge profile for the given EV and the *ISOC* and *FSOC*, the charge duration, *T_ch*, is calculated. In order to calculate *T_ch*, the charge profile data of lithium-ion batteries of the Nissan Altra from Madrid *et al.* [75] is used. If *T_ch* is greater than the duration of the stay of the car (6.1), the algorithm rejects the plan and notifies the PGAM to inform the user of the impossibility of charge completion within the supplied parameters.

$$T\_ch \leq t\_dep - T\_buff - t\_arr \quad \textbf{(6.1)}$$

Otherwise the algorithm calculates the number of charge intervals required for charging, *N_C*, from *T_ch*. The algorithm chooses *N_C* charge intervals (C_i) with the lowest P_j in the interval *(t_arr, t_dep – T_buff)* to charge the EV (Figure 6.8).

6.3.1.2 V2G Operation

If the owner has opted to participate in V2G, the algorithm chooses *N_C' C_i'* intervals with the lowest P_j to charge the EV and *N_D' D_i'* intervals with the highest P_j to send energy from the

EV back into the grid (i.e. sell excess charge at the highest possible price) for maximum profit (Figure 6.8).

$$\sum_{1}^{N\_C'} C_i' - \sum_{1}^{N\_D'} D_i' = 0 \quad \textbf{(6.2)}$$

$$N\_C' = N\_D' \quad \textbf{(6.3)}$$

Purchasing additional charge at cheap time intervals and selling them at higher priced time intervals would generate net profit. It must be noted that the V2G based additional charge and discharge intervals (C_i' and D_i') are equal such that when the EV owner departs, the SOC of his battery is *FSOC* (6.2, 6.3). In scheduling charging and discharging for V2G operation the algorithm must ensure the vehicle's SOC never exceeds 100% or goes below 0% (6.4).

$$\begin{aligned} & t\_arr \le t < t\_dep - T\_buff \\ & 0 \le SOC(t) \le 100 \end{aligned} \quad \textbf{(6.4)}$$

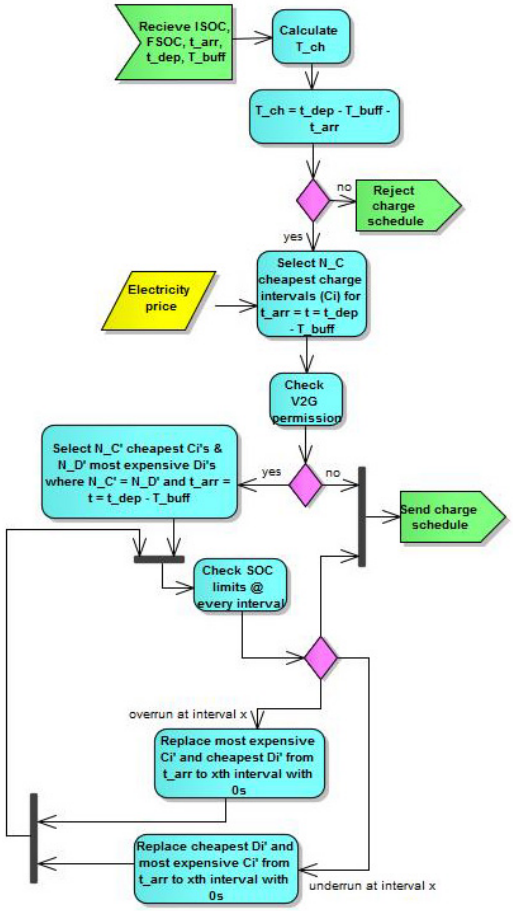

Figure 6.8 EV charging algorithm flowchart.

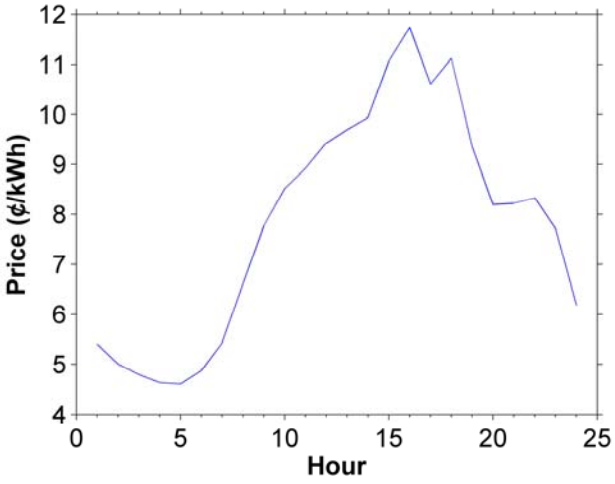

Figure 6.9 Electricity price [76].

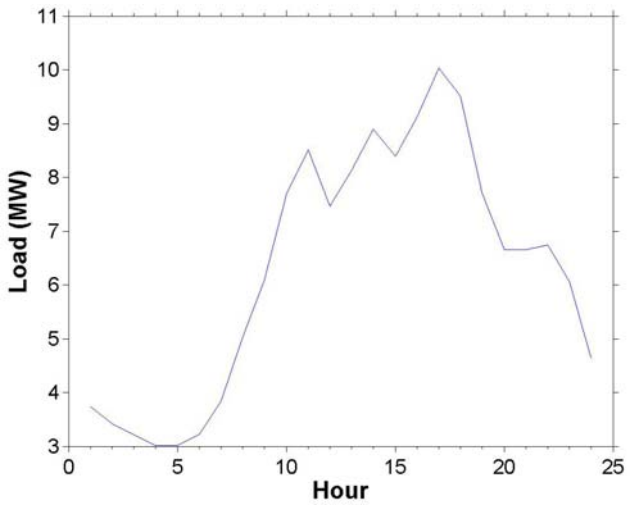

Figure 6.10 Actual Load [76].

6.4 Results

The goals of scheduled charging optimized for electricity price are exploiting off-peak charging benefits and avoiding charging during peak load hours. In addition, while vehicles are parked and idle their energy storage capacity is utilized to alleviate grid load during peak demand. The achievement of this goal is quantified.

6.4.1 Simulation Setup

In order to validate the algorithm, simulations were run using actual day-ahead electricity prices (Figure 6.9), assuming all chargers deliver 6.6 kW, assuming all cars are Nissan Altras and have the charge profile given by Madrid et al.[75], and using two EV driver scenarios: 1) variable schedule and charging requirements 2) enterprise commuter schedule and charging requirements.

Charge scheduling without V2G was simulated using a hypothetical parking garage with 10 chargers and 30 vehicles using the variable scenario. Cost and power usage results are compared with unmanaged charging for the same scenario.

The optimal charge interval duration for maximum V2G profit is determined. Maximum V2G profit is simulated for both types of EV owners.

6.4.2 Charge Scheduling

Charging is scheduled during the cheapest intervals an EV is parked. Comparing the price curve (Figure 6.9) with the charging schedule for 30 variable scenario cars (Figure 6.11) it is obvious that between intervals 10 and 20, when the price is greater than its median value of 8.2 ¢/kWh, charging is minimal. Those cars that are being charged during that interval have time constraints that limit them from being charged at any other time. The V2G schedule for the same 30 cars (Figure 6.12) shows a similar dearth of charging from approximately interval 10 to 20. Most of the discharging happens during this interval when electricity price and demand is highest.

Scheduled charging is more cost-effective than unmanaged charging. The average total cost over 1000 trials of scheduled charging of 30 variable scenario vehicles entering a parking lot of 10 chargers over a 24 hour period was $2.77. Unmanaged charging, which is defined as charging the vehicle as soon as it parks, of the same vehicles cost $3.07. The savings for the variable scenario was 10%. The same simulation was run for the enterprise commuter scenario with 30 vehicles using 30 parking spots instead of 10, to accommodate all the overlapping vehicles. Total average cost was $8.45 when their charging was unmanaged versus $7.87, a savings of 7%.

Scheduled charging also reduces load during peak demand. For the load curve (Figure 6.10), we define peak load as the interval from 11 AM until 7 PM, when the load is higher is higher than its median from 7 AM until 9 PM. Over 1000 trials for the enterprise commuter scenario, unmanaged charging uses an average of 46.5 kW during the peak load interval versus

24.9 kW for scheduled charging – a reduction of 46%. For the variable scenario, unmanaged charging uses an average of 18 kW during peak load versus 7.89 kW for scheduled charging – a reduction of 56%.

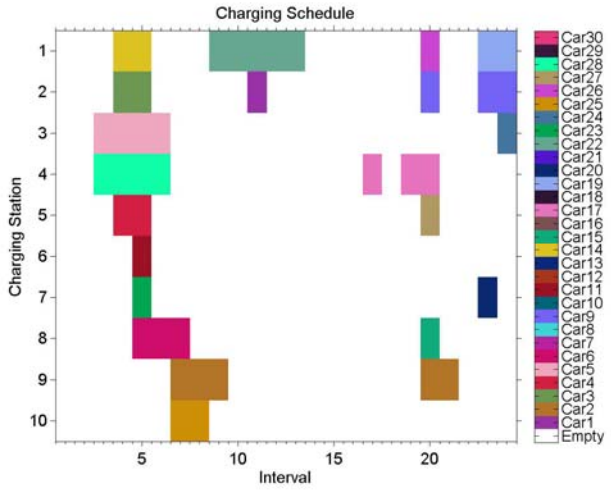

Figure 6.11 Charging schedule for 30 cars, 10 chargers.

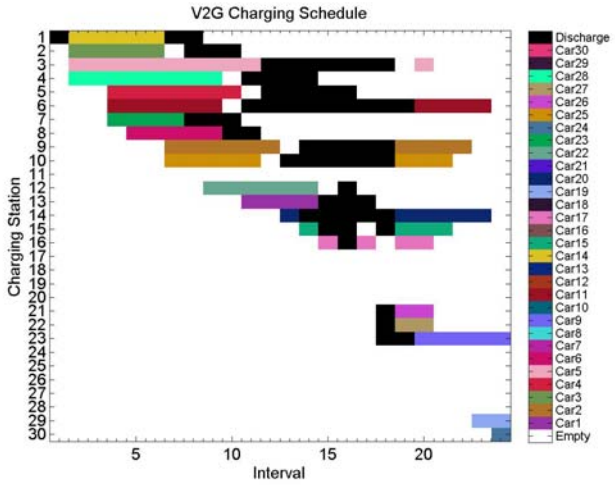

Figure 6.12 V2G schedule for 30 cars, 10 chargers.

6.4.3 V2G Profit

V2G services exploit vehicles' idle time to provide an energy resource during times of peak load. The parking garage operator can incentivize EV owners to participate in V2G using the profits

earned by providing V2G services. Profits are earned when vehicles are charged during off-peak times then send their stored energy back to the grid when demand is high. The effectiveness of V2G services is measured by profit per car, not including the cost of charging to fulfill owner charge profile requirements.

V2G profit per car based on the number of incoming cars and charging stations is shown for incoming cars under the variable (Figure 6.14) and enterprise commuter scenarios (Figure 6.15). It depends on the number and variability of incoming cars, number of charging stations in the parking structure, and the electricity price curve. Each car in the variable car scenario has a random initial arrival time, *ISOC* and random *FSOC* and departure time that are greater than their respective initial counterparts. In the enterprise commuter scenario, arrival times are evenly distributed from 7 AM to noon, departures are evenly distributed from 4 to 9 PM, *ISOC* is log-normally distributed with a mean of 22.3 and a standard deviation of 12.2 [77], and *FSOC* is fixed at 100%.

One hour was determined to be the charge interval duration that maximized V2G profit per car (Figure 6.13). Cars used in determining this value were given a random *ISOC*, *FSOC*, arrival time, and departure time. The test was run with 10 chargers, 100 cars, and averaged over 1000 trials.

The difference between the *FSOC* and *ISOC* variables determines how much time is required to charge the client car – the charging time is not included in V2G profit calculations. However, longer charging times reduce V2G profits by reducing the time available for V2G. Also since cars are used as energy storage, arrival and departure times limit when V2G can occur for each car. V2G profits increase as the ratio of parking duration to required charging time

increases. In order to determine the maximum V2G profit per car, contour plots were generated for the variable scenario (Figure 6.14) and enterprise commuter scenario (Figure 6.15), showing V2G profit per car as a function of number of EVs and number of charging stations. The number of incoming cars and charging stations was varied from 1 to 1001 in increments of 50. Each figure plots 441 different combinations of incoming car and station numbers run over 1000 iterations. A charge interval duration of one hour and buffer time of zero were used. Each contour plot has two regions separated by a saturation limit line. This line represents the car to station ratio where every incoming car undergoes V2G and every station is utilized. The slope of the line demarcating the saturation limit is dependent on the variability of the incoming cars.

The variable incoming car scenario has a saturation limit slope that is greater than one, where the car to station ratio increases with the number of incoming cars. Since there are no constraints on the entry and departure times of incoming cars it is possible for a charging station to accommodate more than one car. Also, as the pool of incoming cars increases, there is an increased likelihood of a car with a later arrival time to fit (park) into a previously occupied charging station after the previous car has left. Below and to the right of this saturation limit line is the charging station overcapacity region where every incoming car is able to park, charge, and go through V2G optimization. Increasing the number of charging stations or decreasing the number of cars in this area will have no affect on the average V2G profit per car, which is 6.9 cents. This particular profit value is a function of the electricity price curve and would vary as the curve varies. In this region, every incoming car is accommodated by an available charging station and thus the V2G profit per car is representative of the entire car population. For example 1 car at 1 parking station over many iterations will have the same mean V2G profit per car as

200 cars at 1000 stations for a given electricity price curve; as they will have similar optimized charge-discharge schedules, giving similar profits. Above the saturation limit line is car oversaturation region where there are insufficient charging stations to accommodate every incoming car. Unlike the overcapacity region, increasing the number of cars or decreasing the number of charging stations in the car oversaturation region significantly affects the V2G profit per car calculation as discussed previously. Maximum profit per car over the entire contour area is 11.7 cents.

For enterprise commuter cars, the latest possible arrival time is noon and the earliest an occupied station is free is 4 PM. Because of these restrictions, it is impossible to have more than one car per station for a large duration (between noon and 4:00 pm) and thus the saturation limit line has a slope of approximately one. Average V2G profit per car is 5.6 cents – this is lower than the variable scenario because of the fixed *FSOC* requirement of 100%, versus a random *FSOC* greater than *ISOC*. Thus on an average, each enterprise commuter scenario car has a longer charging duration and charging stations have less time available for V2G, leading to lower profit.

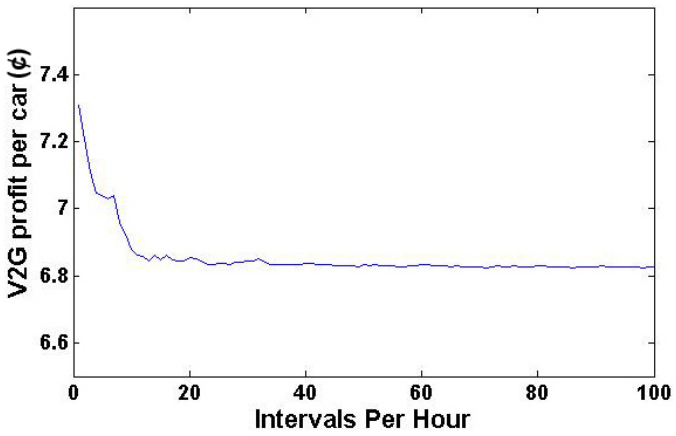

Figure 6.13 V2G profit per car as a function of the number of intervals per hour.

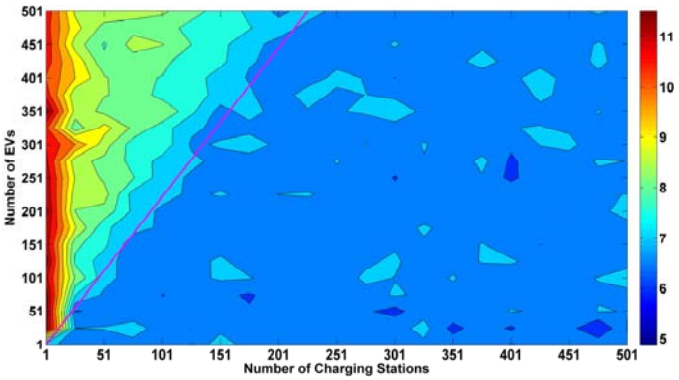

Figure 6.14 V2G profit (¢) contour plot for variable scenario.

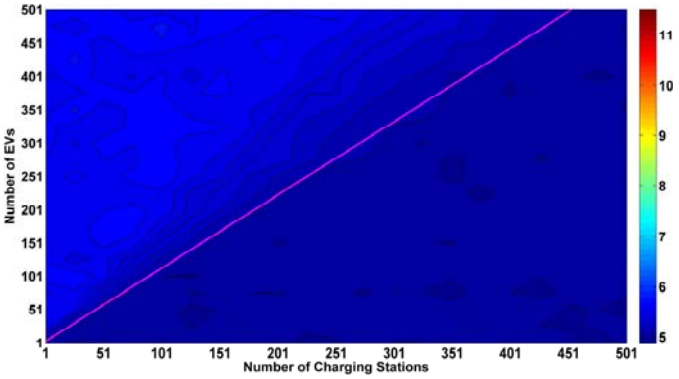

Figure 6.15 V2G profit (¢) contour plot for enterprise commuter scenario cars.

6.5 EV Smart Charging Effectiveness

A comprehensive system leveraging mobile and RFID technologies, aggregation middleware, and an aggregated charge scheduling algorithm, that effectively schedules charging and V2G operations for cost savings and peak load reduction, has been presented.

Intelligently scheduled charging yields a cost savings of 7% for enterprise commuters and 10% for drivers with variable schedule and charging requirements. Peak load can be reduced by 46% for enterprise commuters and 56% for drivers with variable schedules and charging requirements. V2G services that utilize vehicles' idle time, when they are parked but not charging, can generate a net profit for the parking garage operator. A maximum profit of 11.7 cents per vehicle was determined to be achievable for vehicles under the variable scenario and 5.6 cents per vehicle for enterprise commuters.

The proposed system would be well suited for implementation in an enterprise environment where a large number of EVs could be aggregated to substantially impact peak load alleviation and act as a significant energy resource [83].

6.6 Demand Response

Demand Response is defined as "Changes in electric usage by end-use customers from their normal consumption patterns in response to changes in the price of electricity over time, or to incentive payments designed to induce lower electricity use at times of high wholesale market prices or when system reliability is jeopardized" [78, 79]. Aggregated EV charging is an ideal candidate for Demand Response (DR) implementation due to its characteristics of 1) predictable

and controllable load 2) potentially significant load impact 3) fast response time [81]. The existing EV smart charging system architecture can support human-in-the-loop as well as automated DR with no hardware additions and minimal software revision.

6.6.1 System Architecture

Every EVSE sends sensor data via Zigbee or other wireless protocol to an internet-connected base station which then sends it to the PGAM (Figure 6.16). Utility employees create a subscription detailing how often they want power consumption updates and threshold, time value combinations for "Load Exceeded" alerts. When criteria for the subscription is met, and alert is pushed to the user's mobile device and he may act on it by requesting a certain load shedding value during a specified interval. The PGAM relays this information to the ACS which handles rescheduling of EV charging. Another scenario is if the utility company sends a "Curtailment Signal" [80] to the PGAM which it relays to the appropriate users according to their geographical subscription. The user responds with a load shedding request that is sent to the ACS.

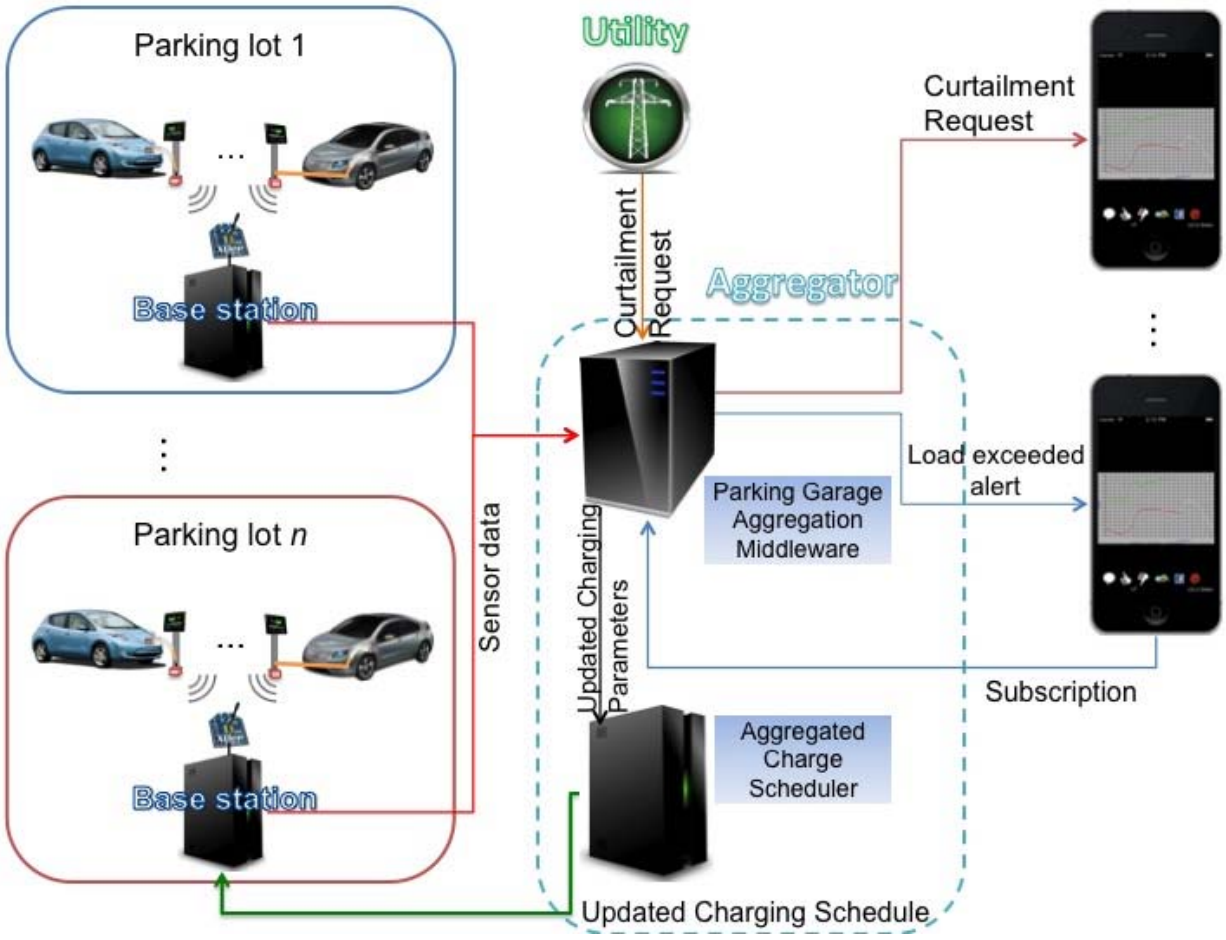

Figure 6.16 DR system architecture.

6.6.1.1 Demand Response Mobile Portal

The Demand Response Mobile Portal (DRMP) is a mobile application that allows a user to 1) select a specific parking lot and shed power during a specified interval (Figure 6.17) 2) select a region and specify an interval and a power constraint and be presented with possible single lots and lot combinations where the constraints can be met.

In the first scenario the user is presented with the current schedule and power consumption for a given lot, and then enters the following parameters: lot number, desired consumption power, and interval beginning and end times (Figure 6.17). The constraints are sent

to the ACS via the PGAM and a rescheduling algorithm is run. If the time and power constraints can be met, a new schedule is created for the lot and is presented to the user and updated commands are sent to the EVSE. If constraints can be partially met, a “best effort” new schedule is sent to the user and ACS (Figure 6.18).

Charger ID List	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	0	0	0	1722	1722	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1721
2	0	1732	1732	1732	1732	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1725
3	0	0	1720	1720	1720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1738
4	0	0	0	1723	1723	0	0	0	0	1723	1714	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1727	1727	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1710	0
6	0	0	0	1737	1737	0	0	0	0	0	0	0	0	0	0	0	0	0	1718	0	0	0	0	0
7	0	0	0	1735	1735	0	0	0	0	0	0	0	0	1735	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1716	0	0	0	0	0	0	0	0	0	0	0	0	0	1726	0	0	0	0	0

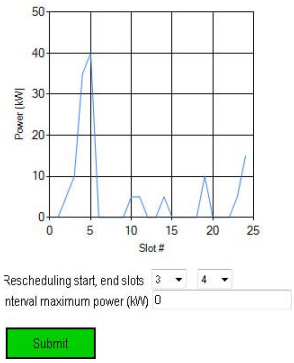

Figure 6.17 DR portal - single parking lot schedule and power consumption.

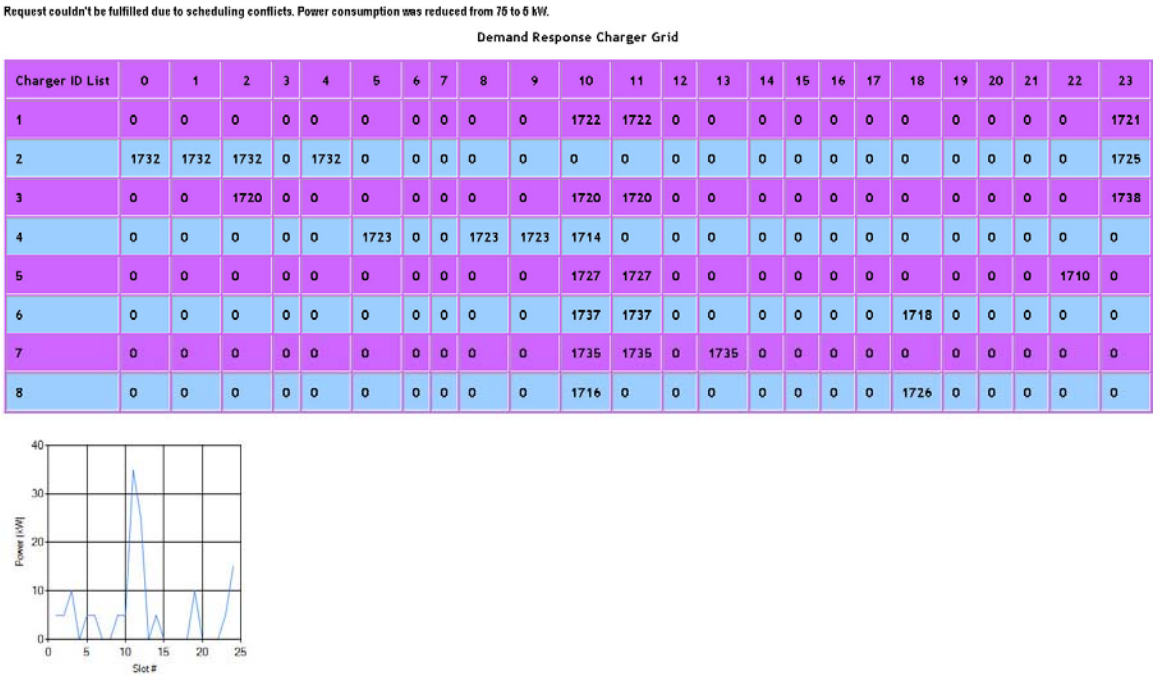

Figure 6.18 DR portal – post curtailment schedule and consumption.

In the second scenario the user selects a geographical region in which to shed power, and then specifies desired consumption power, and interval beginning and end times. The ACS returns to the user all possible single and combinations of lots that can satisfy the constraints in descending order of cost savings.

6.6.2 System Performance

6.6.2.1 Simulation Setup

In order to measure DR notification latencies, simulations were run using real world situations. EVSE power consumption data was simulated for two scenarios: 1) 5 parking lots, located as shown in Figure 6.19, with 20 EVSE each 2) 100 parking lots, 5 located as shown in Figure Lot Locations and the other 95 randomly located, with 10 EVSE each. Both scenarios had a data update period of 2 seconds. One client was connected whose subscription contained a power

consumption threshold of 50 kW, and a location condition given by the upper left and lower right latitude longitude pair (34.068561,-118.448222), (34.067876,-118.445401). A DR notification was dispatched to the client whenever the location was met and power threshold was exceeded. The latency between receipt of the data by the server and receipt of the client's acknowledgement message was measured.

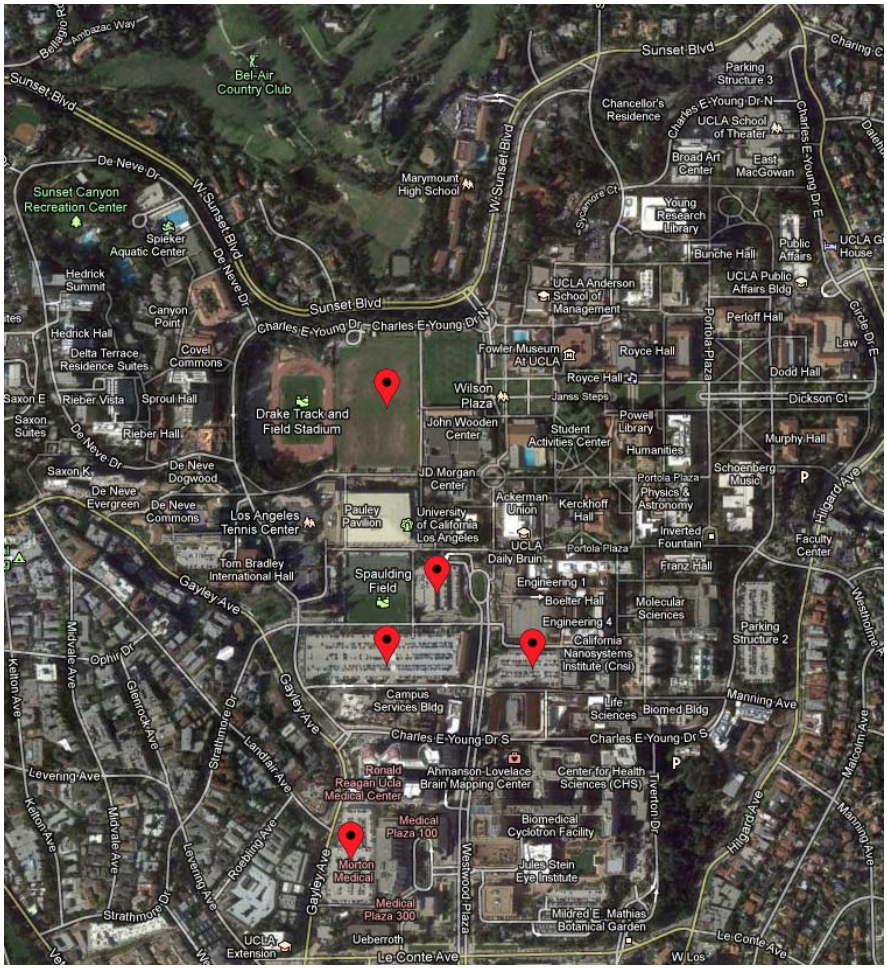

Figure 6.19 Lot locations

6.6.2.2 Simulation Results

For the first scenario, with 5 parking lots and 20 EVSE for a total of 100 data providers, the mean DR notification latency was 95.3 milliseconds (Figure 6.20). The oscillatory characteristic of the latency is due to the nature of the simulated data updates – when data updates from different lots coincided, the server had more data to process and latencies increased. Mean notification latency for the second scenario, with 100 parking lots and 10 chargers per lot for a total of 1000 data providers, was 226.9 milliseconds (Figure 6.21). These results demonstrate that while processing large amounts of sensor data, the PGAM can process and dispatch DR notifications in near real-time.

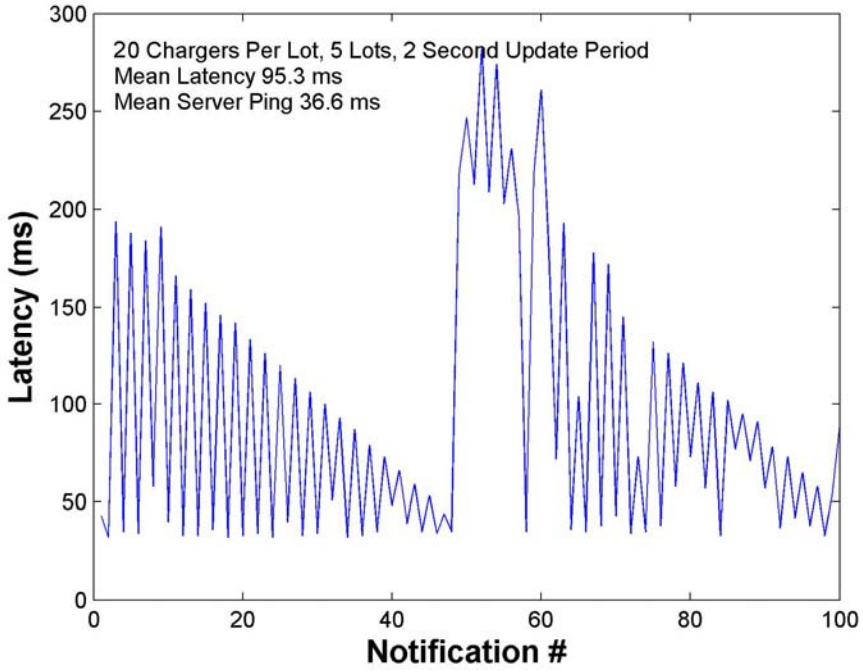

Figure 6.20 DR latency scenario 1

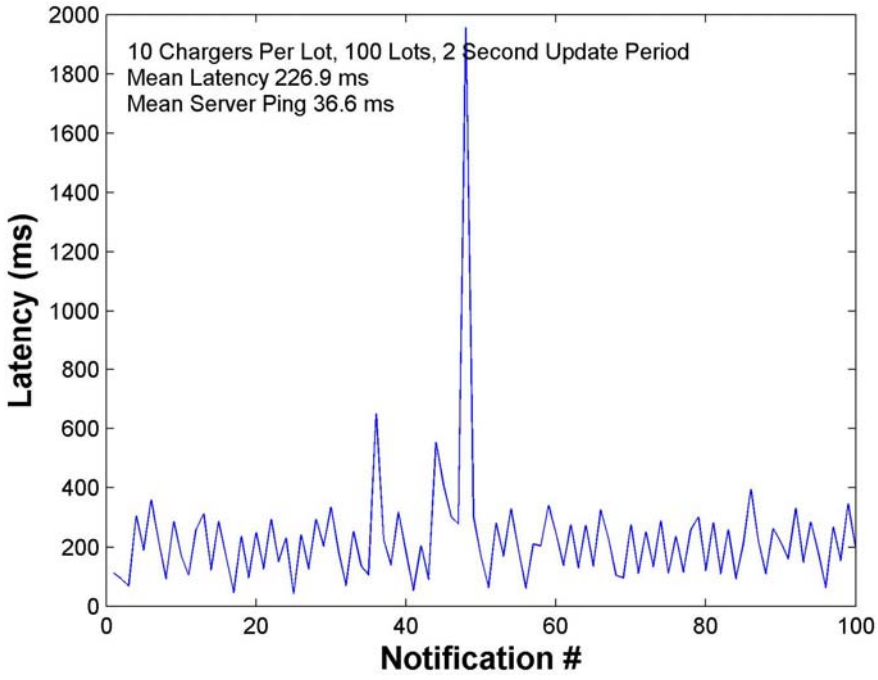

Figure 6.21 DR latency scenario 2

Chapter 7 Conclusion

A near real-time push middleware is developed and shown to enable multimedia content to be aggregated and pushed in near real-time to mobile devices based on their subscription criteria and context in an appropriately adapted format. Performance was quantified for the individual components including the client-server push architecture, the spatial indexing algorithm, and the mobile client architecture. The push architecture demonstrated low latency notifications to clients utilizing a low-overhead messaging scheme to minimize data transferred and maximize mobile client battery life. The spatial indexing algorithm was shown to perform content and subscription searches in less than one millisecond – up to three orders of magnitude lower than traditional methods. The mobile client architecture proved capable of maintaining the server push socket connection while traversing 2G, 3G, and WiFi networks, while maximizing device battery life.

A pilot study of the MobiSportsLive platform was conducted, in which the middleware aggregated and disseminated live video content and user generated content. MobiSportsLive delivered content to mobile clients with 93% lower latency compared to TuVista, an existing research project. In addition, it was able to push new content notifications to mobile clients across a variety of networks, whereas TuVista employed the traditional polling method.

The middleware served as the core of the electric vehicle smart-charging and demand response project, where it aggregated user charging profiles, charger sensor data, alert subscriptions, and demand curtailment signals and pushed vehicle status updates and demand response alerts. Its fast location search algorithm and push architecture enabled multimedia

content and data indexing, storage, push delivery, and subscription matching, that in turn supported smart charge scheduling and demand response alerts and load curtailment.

The significant contributions of this work, the push and mobile architectures, and the spatial indexing algorithm, advanced the state of the art and addressed deficiencies in existing research. MobiSportsLive and electric vehicle smart-charging and demand response each presented novel uses of the developed middleware to solve current research problems. It can be concluded from simulation results and real-world pilot studies that the middleware and its applications consummately address the requirements of industry and academic research.

References

1. Hauswirth, M., Jazayeri, M., A component and communication model for push systems. in *Proceedings of Software Engineering — ESEC/FSE'99. vol. 1687*, pp. 20–28, 1999.

2. Deering, S.E., Cheriton D.R., Multicast routing in datagram inter-networks and extended LANS. *ACM Transactions on Computer Systems*, *8(2)*, pp. 85-110, May 1990.

3. Albanna, Z., Almeroth, K., Meyer, D., Schipper M., IANA Guidelines for IPv4 Multicast Address Assignments. *IETF RFC 3171*, August 2001.

4. Muhl, G., Ulbrich, A., Herrman, K., Disseminating information to mobile clients using publish-subscribe. *Internet Computing*, *IEEE*, *8(3)*, pp. 46- 53, May-Jun 2004.

5. Su, X., Prabhu, B.S., Chu, C.C., Gadh, R., Middleware for Multimedia Mobile Collaborative System. *Wireless Telecommunications Symposium*, 2004, pp. 112- 119, May 14-15 2004.

6. Sacramento, V., Endler, M., Rubinsztejn, H. K., Lima, L. S., Goncalves, K., Nascimento, F.N., Bueno, G.A., MoCA: A middleware for developing collaborative applications for mobile users. *Distributed Systems Online*, *IEEE* , *5(10)*, pp. 2, Oct 2004.

7. Myers, B., Using hand-Held Devices and PCs Together. *Communications of the ACM*, *44(11)*, pp. 34-41, November 2001.

8. Fox, G., Ko, S.H., Kim, K., Oh, S., Lee, S., Integration of Hand-Held Devices into Collaborative Environments. in *Proceedings of the 2002 International Conference on Internet Computing (IC-02)*, pp. 231-250, 2002.

9. Mo, J., Chu, C.C., Prabhu B.S., Gadh R., On the Creation of a Unified Modeling Language based Collaborative Virtual Assembly / Disassembly System. in *Proceedings of the ASME 8th Design for Manufacturing Conference*, Sep 2-6, 2003, Chicago, USA, pp. 273.

10. Shyamsundar, N., Gadh, R., Internet-based Collaborative Product Design with Assembly Features and Virtual Design Spaces. *Computer-aided Design*, *33(9)*, pp. 637-651, August 2001.

11. Jerstad, I., Dustdar, S., Thanh, D.V., A service oriented architecture framework for collaborative services. in *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise,* June 13-15 2005, Linköping, Sweden, pp. 121-125.

12. Balabanovic, M., Shoham, Y., Fab: content-based collaborative recommendation. *Communications of the ACM*, *40 (3)*, pp. 66-72, 1997.

13. Gribble, S.D., Welsh, M., von Behren, R., Brewer, E.A., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, A., Katz, R.H , Mao, Z.M, Ross, S., Zhao, B., The Ninja architecture for robust internet-scale systems and services. *Computer Networks*, *35(4)*, pp. 473-497, March 2001.

14. Armstrong, T., Trescases, O., Amza, C., de Lara, E., Efficient and Transparent Dynamic Content Updates for Mobile Clients. in *Proceedings of the Fourth International Conference on Mobile Systems, Applications and Services*, June 19-22, 2006, Uppsala, Sweden, pp. 56-68.

15. Bhide, M., Deolasee, P., Katkar, A., Panchbudhe, A., Ramamritham, K., Shenoy, P., Adaptive push-pull: disseminating dynamic Web data. *Computers, IEEE Transactions on*, *51(6)*, pp.652-668, June 2002.

16. Rahmati, A., Zhong L., Context-for-wireless: Context-sensitive energy-efficient wireless data transfer. in *Proceedings of the Fifth International Conference on Mobile Systems, Applications and Systems*, June 2007, San Juan, Puerto Rico, pp 165-178.

17. Franks J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Sink, E., Stewart, L., HTTP authentication: Basic and digest access authentication. *IETF RFC 2617*, June 1999.

18. Prabhu, B. S., Su, X., Ramamurthy, H., Chu C.C., Gadh, R., (2006). WinRFID - A Middleware for the enablement of Radio Frequency Identification (RFID) based Applications. *in Mobile, Wireless, and Sensor Networks: Technology, Applications, and Future Directions* (eds R. Shorey, A. L. Ananda, M. C. Chan and W. T. Ooi), John Wiley & Sons, Inc.

19. Podnar, I., Hauswirth, M., Jazayeri, M., Mobile Push: Delivering Content to Mobile Users. in *Proceedings of the First International Workshop on Distributed Event-Based Systems*, July 2-3, Vienna, Austria, pp. 563-570.

20. Zhou, Y., Xie, X., Wang, C., Gong, Y., Ma, W., Hybrid index structures for location-based web search. in *Proceedings of the 14th Conference on Information and Knowledge Management*, October 31 – November 5, Bremen, Germany, pp. 155-162.

21. Sacramento, V., Endler, M., Nascimento, F.N., A Privacy Service for Context-aware Mobile Computing. in *Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks*., September 5-9, 2005, Athens, Greece, pp. 182- 193.

22. Kurose, J. F., Ross, K. W. (2010). *Computer Networking: A Top-Down Approach* (5th ed.). Pearson Education.

23. Ford, B., Srisuresh, P., Kegel, D. Peer-to-peer communication across network address translators. In USENIX (2005).

24. Hitchens, R. (2002). *Java NIO* (1st ed.). O'Reilly and Associates, Inc.

25. Perkins, C.E., Mobile IP. *Communications Magazine*, *IEEE , 35(5)*, pp.84-99, May 1997.

26. Lee, J.H., Jung, T.H., Yoon, S.U., Youm, S.K., Kang, C.H., An adaptive resource allocation mechanism including fast and reliable handoff in IP-based 3G wireless networks. *Personal Communications, IEEE*, *7(6)*, pp. 42-47, December 2000.

27. Lach, H.Y., Janneteau, C., Petrescu, A., Network mobility in beyond-3G systems. *Communications Magazine, IEEE*, *41(7)*, pp. 52- 57, July 2003.

28. Rogerson, P.A., The Detection of Clusters Using a Spatial Version of the Chi-Square Goodness-of-Fit Statistic. *Geographical Analysis*, 31(2), pp. 130–147, April 1999.

29. Josefsson, S., The Base16, Base32, and Base64 Data Encodings. *IETF RFC 3548*, July 2003.

30. Apache Commons. (March 2012). The Apache Commons Mathematics Library, http://commons.apache.org/math/, March 2012.

31. Ramakrishnan, R., Gehrke. J. (2002). *Database Management Systems* (3rd ed.). McGraw-Hill.

32. Cormen, T. H., Leiserson, C. E., Rivest, R. L. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

33. Ramamurthy, H., Karandikar, A., Gadh, R. (2003). Efficient Broadband Multimedia data distribution over the Internet using Multicast, http://www.wireless.ucla.edu/techreports/UCLA-WINMEC-2003-609-MULTICAST-BBAND.pdf, March 2012.

34. W3C. (August 16, 2011). SVG specification, http://www.w3.org/TR/SVG/, March 2012.

35. Su, X., Prabhu, B.S., Gadh, R. (2003). Mobile Internet for the Multimedia Enterprise (MobIME), http://www.wireless.ucla.edu/techreports/UCLA-WINMEC-2003-303-Mobile-Multimeida-EN.pdf, March 2012.

36. Maggiorini, D., Cazzola, W., Prabhu, B.S., Gadh, R. (2003). A service Oriented Middleware for Seamless System Nomadic-Aware (SNA) Servants, http://www.wireless.ucla.edu/techreports/UCLA-WINMEC-2003-302-LDIS-SNA.pdf, March 2012.

37. Su X., Prabhu B.S., Chu C.C., Gadh R., Middleware for Multimedia Mobile Collaborative System, in *Proceedings of the Wireless Telecommunications Symposium 2004*, Cal Poly Pomona, Pomona, CA, USA, May 14-15, 2004, pp. 112-119.

38. Ramamurthy, H., Lai, D., Prabhu, B.S., Gadh, R., ReWINS: a distributed multi-RF sensor control network for industrial automation. in *Proceedings of the Wireless*

Telecommunications Symposium 2005, Cal Poly Pomona, Pomona, CA, USA, April 28-30, 2005, pp. 24- 33.

39. Kumar P., Sridhar G., Sridhar V., Gadh R., DMW- A Middleware for Digital Rights Management in Peer-to-Peer Networks. in *Proceedings of the 16th International Workshop on Database and Expert Systems Applications*, Copenhagen, Denmark, August 22 - 26, 2005, pp. 246-250.

40. Chu, C.C., Su, X., Prabhu B.S., Gadh, R., Kurup, S., Sridhar, G., Sridhar, V., Mobile DRM for multimedia content commerce in P2P networks. in *Proceedings of the Third Consumer Communications and Networking Conference, IEEE*, Las Vegas, NV, USA, January 8-10, 2006, pp.1119-1123.

41. Su X., Prabhu B.S., Chu C.C., Gadh R., Scalable Vector Graphics (SVG) Based Multi-Level Graphics Representation for Engineering Rich-Content Exchange in Mobile Collaboration Computing Environments. *Journal of Computing and Information Science in Engineering*, *6(2)*, pp. 96-102, June 2006.

42. Chiang K., Prabhu B.S., Chu C.C., Gadh R., In the Direction of a Sensor Mapping Platform Based on Cellular Phone Networks. in *Proceedings of the Wireless Telecommunications Symposium 2008*, Cal Poly Pomona, Pomona, CA, USA, April 24-26, 2008, pp. 348-353.

43. Park, N., Gadh, R., Implementation of Cellular Phone-based Secure Light-Weight Middleware Platform for Networked RFID. in *Proceedings of the International Conference on Consumer Electronics 2010*, Las Vegas, NV, USA, January 9-13, 2010, pp.495.

44. Wireless Communications and Public Safety Act of 1999, S. 800, 106h Cong. (1999).

45. Zhang, J., Li, B., Rizos, C., Dempster, A.G., Evaluation of high sensitivity GPS receivers. in *Proceedings of the International Symposium on GPS/GNSS 2010*, Taipei, Taiwan, October 26-28, 2010, pp. 410-415.

46. Wang, S., Min, J., Yi, B.K., Location Based Services for Mobiles: Technologies and Standards. in *Proceedings of the International Conference on Communication 2008, IEEE*, Beijing, China, May 19-23, 2008.

47. Miluzzo, E., Oakley, J.M.H., Lu, H., Lane, N.D., Peterson, R.A., Campbell, A.T., Evaluating the iPhone as a mobile platform for people-centric sensing applications. in *Proceedings of the International Workshop on Urban, Community, and Social Applications of Networked Sensing Systems 2008*, Rayleigh, NC, USA, November 4, 2008, pp. 41-46.

48. Shu, X., Du, Z., Chen, R., Research on Mobile Location Service Design Based on Android. in Proceedings of the *Fifth International Conference on Wireless Communications, Networking and Mobile Computing, 2009*., Beijing, China, September 24-26, 2009, pp.1-4.

49. Su, X., Gadh, R., A Rule Language and Framework for RFID Data Capture and Processing in Manufacturing Enterprise System. *International Journal of Internet Manufacturing and Services*, *2(2)*, pp. 111-127, February 2010.

50. Su, X. (2003) "WinRFID – A Middleware for the enablement of Radio Frequency Identification based Applications" (Doctoral thesis).

51. Boag, S., Chamberlin, D., Fernandez, M., Florescu, D., Robie, J., Simeon, J. (2005) 'XQuery 1.0: An XML query language' W3C Candidate Recommendation. http://www.w3.org/TR/2005/CR-xquery-20051103/, March 2012.

52. Boley, H., Tabet, S., Wagner. G., Design Rationale of RuleML: A Markup Language for Semantic Web Rules. in *Proceedings of the First Semantic Web Working Symposium 2001*, Stanford, CA, USA, July 30 – August 1, pp. 381–401.

53. United States Department of Energy. (February 2011) "One Million Electric Vehicles By 2015," http://energy.gov/sites/prod/files/edg/news/documents/1_Million_Electric_Vehicle_Report_Final.pdf, March 2011.

54. Kiviluoma, J., Meibom, P., Methodology for modelling plug-in electric vehicles in the power system and cost estimates for a system with either smart or dumb electric vehicles. *Energy*, 36(3), pp. 1758-1767, March 2011.

55. Kintner-Meyer, M., Schneider, K., Pratt, R., Impacts assessment of plug-in hybrid electric vehicles on electric utilities and regional U.S. power grids. Department of Energy, Pacific Northwest National Laboratory (PNNL). Contract DE-AC05-76RL01830. December 2006.

56. Kempton, W., Tomić, J., Vehicle-to-grid power implementation: From stabilizing the grid to supporting large-scale renewable energy. *Journal of Power Sources*, *144(1)*, pp. 280-294, June 1, 2005.

57. Guille, C., Gross, G., A conceptual framework for the vehicle-to-grid (V2G) implementation. *Energy Policy*, 37(11), pp. 4379-4390, November 2009.

58. Ferreira, J., Monteiro, V., Afonso, J., Silva, A. Smart Electric Vehicle Charging System. in *Proceedings of the 2011 Intelligent Vehicles Symposium, IEEE*. Baden-Baden, Germany, June 5-9, 2011, pp. 758-763.

59. Shao, S., Pipattanasomporn, M., Rahman, S., Challenges of PHEV penetration to the residential distribution network. in *Proceedings of the 2009 Power & Energy Society General Meeting, IEEE*, Calgary, AB, Canada, July 26-30, 2009, pp.1-8.

60. Yu, X. Impacts assessment of PHEV charge profiles on generation expansion using national energy modeling system. in *Proceedings of the 2008 Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, IEEE*, Pittsburgh, PA, USA, July 20-24, 2008, pp.1-5.

61. Finkenzeller, K. (2003). *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification (*2nd ed.*)*. John Wiley & Sons, Inc.

62. Roberts, C., Radio frequency identification (RFID). *Computers Security*, *25(1)*, pp. 18-26, 2006.

63. Paret, D., Technical state of art of "Radio Frequency Identification -RFID" and implications regarding standardization, regulations, human exposure, privacy. in *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-Aware Services: Usages and Technologies, ACM*, Grenoble, France, October 12-14, 2005, pp. 9-11.

64. Porter, J.D., Kim, D.S., An RFID-Enabled Road Pricing System for Transportation. *Systems Journal, IEEE*, *2(2)*, pp. 248-257, June 2008.

65. Theo, L., Jonas, F., Using the Energy Name Service (ENS) for electric mobility roaming. in *Proceedings of eChallenges 2010*, Warsaw, Poland, October 27-29, 2010, pp.1-7.

66. Song W.C., Authentication System for Electrical Charging of Electrical Vehicles in the Housing Development. in *Proceedings of Security-Enriched Urban Computing and Smart Grid Communications in Computer and Information Science 2010*, Daejeon, Korea, September 15-17, 2010, pp. 261-266

67. Soares, J., Sousa, T., Morais, H., Vale, Z., Faria, P., An optimal scheduling problem in distribution networks considering V2G. in *Proceedings of the 2011 Symposium on Computational Intelligence Applications In Smart Grid, IEEE*, Paris, France, April 11-15, 2011, pp. 1-8.

68. Venayagamoorthy, G.K., Mitra, P., Corzine, K., Huston, C., Real-time modeling of distributed plug-in vehicles for V2G transactions. in *Proceedings of the 2009 Energy Conversion Congress and Exposition, IEEE*, San Jose, CA, USA, September 20-24 2009, pp. 3937-3941.

69. Hutson, C., Venayagamoorthy, G.K., Corzine, K.A., Intelligent Scheduling of Hybrid and Electric Vehicle Storage Capacity in a Parking Lot for Profit Maximization in Grid Power Transactions. in *Proceedings of the Energy 2030 Conference, IEEE,* Atlanta, GA, USA, November 17-18, 2008, pp. 1-8.

70. Wu, D., Chau, K.T., Gao, S., Multilayer framework for vehicle-to-grid operation. in *Proceedings of the Vehicle Power and Propulsion Conference, IEEE*, Lille, France, September 1-3, 2010, pp.1-6.

71. Saber A.Y., Venayagamoorthy, G.K., Intelligent unit commitment with vehicle-to-grid - A cost-emission optimization. *Journal of Power Sources*, *195(3)*, February 1, 2010, pp. 898-911.

72. Schieffer, S.V. (2010). *To charge or not to charge? Decentralized charging decisions for the smart grid.* (Master's thesis).

73. Han, S., Han, S.H., Sezaki, K., Design of an optimal aggregator for vehicle-to-grid regulation service. in *Proceedings of the PES Conference on Innovative Smart Grid Technologies*, *IEEE*, Gaithersburg, MD, USA, January 19-21, 2010, pp. 1-8.

74. Tate, E., Harpster, M., Savagian, P., The Electrification of the Automobile: From Conventional Hybrid, to Plug-in Hybrids, to Extended-Range Electric Vehicles. SAE International Journal of Passenger Cars – Electronic and Electrical Systems, 1(1), pp. 156-166, April 2008.

75. Madrid, C., Argueta, J., Smith, J., Performance characterization - 1999 Nissan Altra-EV with lithium-ion battery. *Southern California EDISON,* Sep. 1999.

76. Xu, Y., Xie, L., Singh, C., Optimal scheduling and operation of load aggregator with electric energy storage in power markets. in *Proceedings of the North American Power Symposium, 2010*, Arlington, TX, USA, September 26-28, 2010, pp.1-7.

77. Qian, K., Zhou, C., Allan, M., Yuan, Y., Modeling of Load Demand Due to EV Battery Charging in Distribution Systems. *IEEE Transactions on Power Systems, 26(2)*, pp. 802-810, May 2011.

78. Albadi, M.H., El-Saadany, E.F., Demand Response in Electricity Markets: An Overview. in *Proceedings of the Power Engineering Society General Meeting, IEEE*, Tampa, FL, USA, June 24-28, 2007, pp. 1-5.

79. U.S. Department of Energy. (February 2006) "Benefits of demand response in electricity markets and recommendations for achieving them," http://eetd.lbl.gov/ea/EMP/reports/congress-1252d.pdf, March 2011.

80. Aalami, H.A., Moghaddam, M.P., Yousefi, G.R., Demand response modeling considering Interruptible/Curtailable loads and capacity market programs. *Applied Energy*, *87(1)*, pp. 243-250, January 2010.

81. Ipakchi, A., Albuyeh, F., Grid of the future. *IEEE Power and Energy Magazine, 7(2)*, pp.52-62, March-April 2009.

82. Mal, S., Gadh, R., Real-time push middleware and mobile application for electric vehicle smart charging and aggregation. *International Journal of Communication Networks and Distributed Systems, Special Issue on: Context-Aware System and Intelligent Middleware for Smart Grid*, Accepted for publication June 15, 2011.

83. Mal, S., Chattopadhyay, A., Yang, A., Gadh, R., Electric vehicle smart charging and vehicle-to-grid operation, *International Journal of Parallel, Emergent and Distributed Systems*, *27(3),* pp. 1-17, May 2012.

84. Bozdag, E., Mesbah, A., van Deursen, A., A comparison of push and pull techniques for Ajax. in *Proceedings of the 9th International Symposium on Web Site Evolution, IEEE*, Paris, France, October 5-6, 2007, pp. 15–22.

85. Junglas, I., Watson, R., Location-based services. *Communications of The ACM*, *51(3)*, pp 65–69, March 2008.

86. Massoud Amin, S., Wollenberg, B.F., Toward a smart grid: power delivery for the 21st century, *Power and Energy Magazine, IEEE*, *3(5)*, pp. 34- 41, September-October 2005.