Organizing and Accessing Web Services on Air

Xu Yang, Athman Bouguettaya, *Senior Member, IEEE*,
Brahim Medjahed, *Student Member, IEEE*, Hao Long, and Weiping He

***Abstract*— Mobile commerce (*m-commerce*) refers to the conduct of business using wireless devices and communications. Driven by the success of e-commerce and impressive progress in wireless technologies, m-commerce is rapidly taking place in the business forefront. However, most of the concepts developed for e-commerce may not be easily applicable to wireless environments. This is due to the peculiarities of these environments such as limited bandwidth, unbalanced client-server communication, and limited power supply. *Web services* are undeniably one of the most significant e-commerce concepts worth of being adapted to the wireless world. Mobile services, also called *m-services*, promise several benefits compared with their wired counterparts. They provide larger customer base and cater for *"anytime and anywhere"* access to services. In this paper, we propose an infrastructure for organizing and efficiently accessing m-services in *broadcast* environments. We define a *multi-channel* model to carry information about m-services available within a given geographic area. The *UDDI channel* includes registry information about m-services. The *m-service* channel contains the description and executable code of each m-service. The *data channel* contains the actual data needed while executing the m-service. We also introduce three techniques to enable efficient access to wireless channels. These techniques extend well-known mobile databases' access methods to m-services: B+ tree, signature indexing, and hashing. We finally present an analytical model and conduct an extensive experimental study to evaluate and compare the proposed techniques.**

I. Introduction

The recent advances in hardware, networking, and Web technologies radically changed the way companies conduct their daily business. While initially aimed at enabling the sharing of information among scientists, the Web has evolved to become a promising medium for *e-commerce* activities [1], [2]. A broad spectrum of e-commerce applications is already available on today's Web such as on-line banking and shopping. However, most of these applications were developed for wired infrastructures with fixed or stationary users [3]. The past years has witnessed a boom in wireless technologies [4]. Sophisticated wireless devices such as cellular phones and PDAs (*Personal Digital Assistants*) are now available at affordable prices. Emerging technologies including 3G and 4G (third and fourth generation) are under development to increase the bandwidth of wireless channels. Driven by the success of e-commerce and impressive progress in wireless technologies, *mobile commerce* (*m-commerce*) is rapidly taking place in the business forefront. M-commerce refers to the conduct of business over wireless communications and devices [5]. Examples of m-commerce applications include mobile office (e.g., working while on the move) mobile advertising (e.g., location sensitive advertisements), and mobile financial applications (e.g., banking and payment for mobile users) [6].

Enabling technologies for e-commerce have been around for almost three decades. They provide businesses with means for interacting with their peers (B2B e-commerce) and customers (B2C e-commerce). *Web services* are undeniably one of the most significant e-commerce technologies. A Web service is a set of related business functions that can be programmatically accessed through the Web [7], [8]. The widespread adoption of Web services has spurred an intense research activity to address Web services issues (e.g., organization, access, composition) [9]. However, most of the proposed techniques dealing with those issues cannot or may not be easily applicable to m-commerce. This is due to the peculiarities of wireless environments including limited bandwidth, unbalanced client-server communication, limited power supply, and frequent unavailability of wireless networks [10]. For example, using UDDI (*Universal Description, Discovery and Integration*) [11] for discovering Web services requires multiple *costly* round trips over wireless networks [12]. Invoking Web services using SOAP (*Simple Object Access Protocol*) [13] may increase mobile hosts' power consumption and waiting time. This calls for new techniques to adapt Web services to the wireless world [14].

To support wireless-oriented services, a new generation of Web services called *mobile services* (*m-services*) has emerged. An m-service is a Web service that is accessible by mobile hosts through wireless networks [3], [12]. An example of m-service is a `stock quote` service providing stock quote prices to users on the move. M-services promise several benefits compared with their wired counterparts. They provide larger customer base. It is expected that by 2004, 50 percent of Internet access in the US will originate from wireless devices [4]. M-services also cater for "*anytime and anywhere*" access to services. Users need no longer to sit in front of their desktop computers to conduct their business transaction.

An important issue in wireless-oriented service environments is *organizing* and *accessing* m-services. Key requirements for accessing m-services include minimizing power consumption and client's waiting time. We identify three modes for accessing m-services: *at-hand*, *on-demand*, and *broadcast*. In the *at-hand* mode, services are pre-installed on users' wireless devices. This mode is clearly inappropriate in open environments where users' requests are unpredictable and the number of m-services is large and dynamic. In the *on-demand* mode, mobile users send their operation invocations along with the requested parameters (e.g., stock name) to the m-service provider (generally in wired network) through an

This research is supported by the National Science Foundation under grant 9983249-EIA and by a grant from the Commonwealth Information Security Center (CISC).

The authors are with the Department of Computer Science, Virginia Tech, 7054 Haycock Road, Falls Church, VA 22043, USA.

Email: {xuyang, athman, brahim, halong, weiping}@vt.edu

uplink channel (client-to-server channel). After executing the operation, the provider replies by sending back the operation's results (e.g., stock price) through a *downlink* channel (server-to-client channel). Since sending data from a wireless device is a power consuming process, this mode may be inefficient for applications where several requests are issued by mobile users (e.g., stock market applications). In the *broadcast* mode, a service broker periodically broadcasts the available m-services over the wireless channel. Clients listen to the channel, identify the m-service of interest, and download it to the mobile host for local execution. Broadcasting avoids power consuming uplink transmissions. It has been demonstrated to save considerable amount of power [15], [16]. Broadcast is suitable for a large number of clients with overlapping interests (as in m-commerce applications). Additionally, performance does not depend on the number of clients. Finally, the server is prevented from being overwhelmed by client requests. In this paper, we propose an infrastructure for organizing and efficiently accessing m-services in broadcast environments. More precisely, this paper's contributions include the following:

- **Multi-channel model for m-services** – We propose a multi-channel model for organizing and accessing m-services. Multiple channels are defined to carry different types of information about m-services available within the same geographical area (cell or registration area). The *UDDI channel* includes registry information about m-services. The *m-service* channel contains the description and executable code of each m-service. The *data channel* contains the actual data needed while executing the m-service in the mobile host.
- **Wireless channels access methods** – Broadcast requires techniques for efficiently locating m-services on the wireless channel. We present three indexing techniques for accessing wireless channels in the proposed m-service architecture. The aim of these techniques is to reduce power consumption and client's waiting time. The proposed techniques extend well-known access methods in mobile databases to m-services: B+ tree, signature indexing, and hashing.
- **Analytical and experimental study** – We present an analytical model of each proposed access method. We have also conducted an extensive experimental study to evaluate and compare those methods.

The remainder of this paper is organized as follows. In Section II, we propose a multi-channel model for m-services. In Section III, we present some techniques for accessing wireless channels. In Section IV, we describe the analytical model for assessing the wireless channels access methods. In Section V, we present the simulation experiments and their results. In Section VI, we give an overview of the related work. We finally provide concluding remarks in Section VII.

II. Proposed Infrastructure for M-services

Two major approaches are possible for executing m-services: *remote* and *local*. By executing an m-service, we mean invoking one of its operations. The *remote* approach, applicable to the on-demand access mode, is similar to the traditional client-server approach. M-services are executed by their providers at the server side. Remote execution is particularly suitable for wireless devices with limited processing capabilities. While this approach is generally adopted in Web services, it may be inappropriate for m-service environments where mobile devices lose network connectivity much more than fixed hosts do in wired networks. In the *local* approach, applicable to at-hand and broadcast access modes, wireless devices act as a computing platform for m-services. This requires *fat* clients with advanced processing capabilities. Such requirement is nowadays technically feasible with the current progress in wireless devices technologies. For example, it is today possible to install Web/application servers and servlet engines in PDAs. It is expected that more powerful wireless devices will be available in the near future [4]. Figure 1 presents a novel architecture to support local execution of m-services in broadcast environments.

Designing an m-service infrastructure for broadcast environments requires an effective organization of the m-service space. Such organization would empower mobile users with the the ability to (1) discover m-services of interest; (2) download the m-service code; (3) find out the way to invoke the m-service (e.g., which inputs are required); and (4) download the data needed during m-service execution. Figure 1 depicts our architecture to deal with the aforementioned issues.

Components – The proposed architecture involves three types of participants: *mobile clients*, *m-service providers*, and *m-service brokers* (Figure 1). Mobile clients are end-users equipped with wireless devices. They have access to m-services that are available within their current *geographic area*. A geographic area may be a cell or a registration area (i.e., group of few cells) as defined in North America's IS-41 (*Interim Standard 41*) standard for wireless communication [17]. M-service providers are the entities that offer business services to mobile clients. Each provider has to subscribe with one of the *brokers* available in its geographic area. During the subscription process, providers need to give registry information and descriptions of their m-services. They also need to provide the code of each m-service and data required during its execution. We assume that m-service codes are simple enough to be runnable by wireless devices. Brokers act as intermediaries between m-service providers and mobile clients. They broadcast information about m-services (registry, description, code, and data) to mobile clients through wireless networks. Since these m-services are available for certain geographic areas, we assume that each broker has the capacity of broadcasting up to a few hundred m-services. This will guarantee the performance of accessing m-services provided by each broker. For areas where only a small number of m-services are available, a single broker may be enough for broadcasting all m-services. For areas with high population density, there may be a large number of m-services provided for users in each area. In this case, m-services can be categorized into different families. Multiple brokers will be used to broadcast these m-services, with one or more broker broadcasting one family of m-services. In this paper, we consider the case of one broker per area.

To illustrates interactions between participants

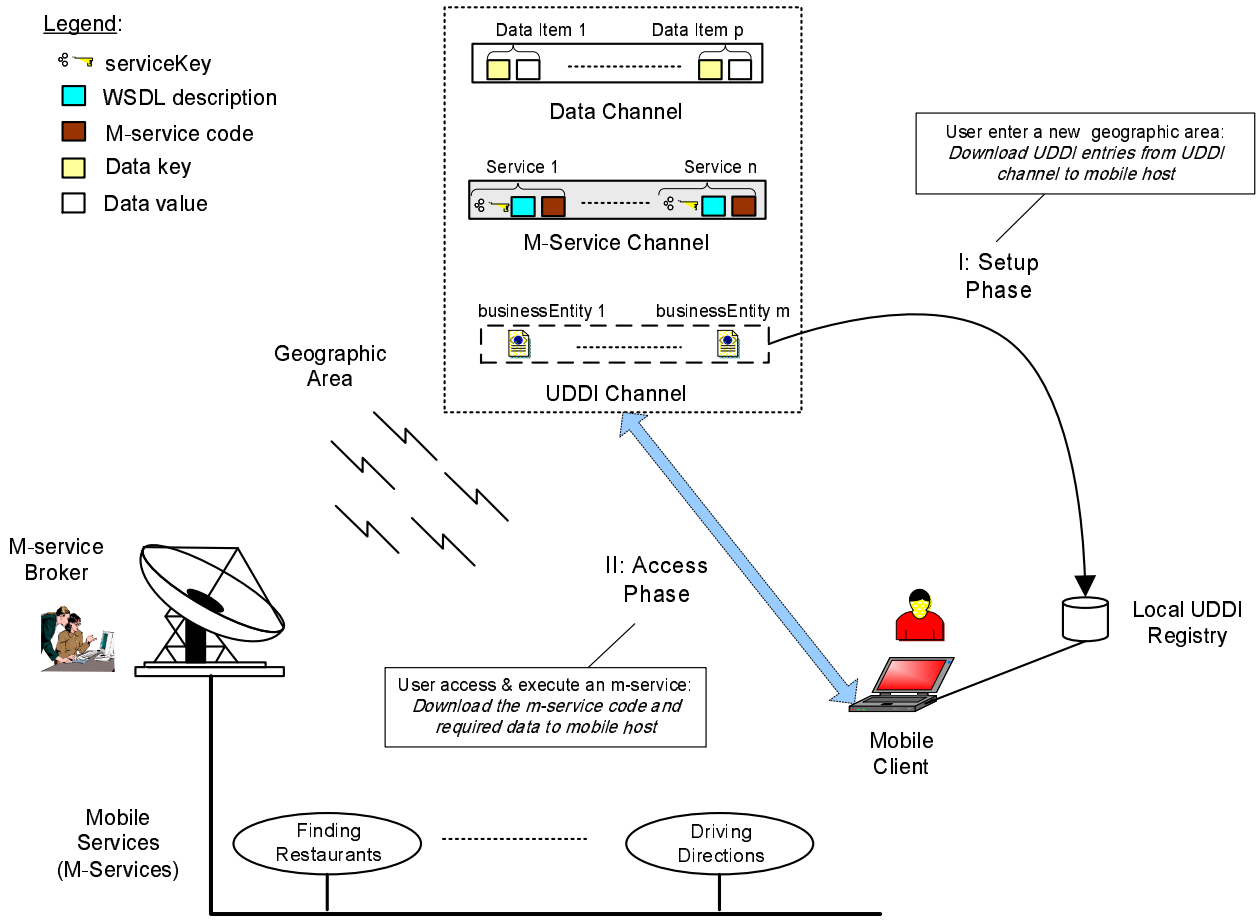

Fig. 1. Multi-channel Based Architecture for M-services

in the proposed architecture, let us consider a `restaurant_finder` m-service. This m-service allows mobile users to find restaurants based on user's location and restaurants' specialties (e.g., Chinese food, Italian food). The m-service provider feeds the broker with the m-service code and list of restaurants available within the local area. This list includes restaurants' names, locations (street address), and phone numbers. Among other information, the broker broadcasts the m-service code and list of available restaurants. This list is used by the m-service code during execution at the wireless device. Such execution involves the following simple steps. The mobile user first gives its current street address and the restaurant specialties. The m-service then compares user's inputs with the broadcast data. If a matching is found, the m-service returns the relevant restaurant names with their addresses and phone numbers.

UDDI Channel – The use of "traditional" UDDI for discovering m-services incurs several costly round trips over wireless networks. To avoid such problems, we associate a *UDDI channel* within each geographic area (Figure 1). The UDDI channel contains the directory information of all m-services provided within that area. This information is given by m-service providers and broadcast by brokers. Each entry in the UDDI channel is identified by a `businessKey`. The information contained in the UDDI channel is customized to fit the characteristics of wireless environments. For example, the `accessPoint` attribute within UDDI's `bindingTemplate` structure no longer contains the HTTP or FTP addresses to access the service [11]. It rather contains the frequency of the wireless channel through which the m-service can be accessed. Additionally, UDDI's `businessService` will contain no URL link to the service description.

As depicted in Figure 1, whenever mobile clients enter a new geographic area (*setup* phase), they download the UDDI channel's content to their mobile device and store it for later use. Caching the directory of m-services in users' devices avoids frequent access to the UDDI channel and hence minimizes power consumption. Note that the number of m-services within a given area is relatively small. Hence, downloading the UDDI registry to the wireless device is technically feasible. To keep the cached UDDI registry updated, a version number is associated to each UDDI channel's broadcast cycle. Before users access the cached UDDI registry, its version number is checked against the broadcast version number. If they are different, a new UDDI registry will need to be downloaded from this channel again.

M-service and Data Channels – Once an m-service is discovered, another important issue concerns service access and execution. To cope with this issue, we associate two other wireless channels to each geographic area: *m-service*

```
Algorithm execute-m-service

   /* executed whenever a mobile client wants to access an m-service */
   /* in the local geographic area */
Begin
   1: find m-services having a given category in the local UDDI registry.
      UDDI inquiry API is used for that purpose.
   2: select an m-service and retrieve its serviceKey K from businessService.
   3: retrieve the frequency of the m-service and data channels.
   4: listen to m-service channel in the local geographic area.
   5: download the description and code of the m-service having K as a
      serviceKey.
   6: select the m-service operation that needs to be invoked.
   7: determine the input parameters required by that operation from the
      m-service description.
   8: execute the m-service code locally.
   9: if the execution requires a set of data D then listen to the data
      channel in the local geographic area.
  10: use primary data key to download all data in D from the data channel.
  11: continue m-service execution at the wireless device.
  12: return results to the user.
 End
```

Fig. 2. Algorithm for Executing M-Services

and *data* channels (Figure 1). Each entry in the first channel contains the executable code and description of a given m-service. For better access performance, information within the m-service channel is indexed using `serviceKey`. A `serviceKey` takes the form of a UUID (*Universally Unique ID*) that uniquely identifies m-services [11]. It is similar to the `serviceKey` defined within a UDDI's `businessService`. The m-service description allows mobile users to find out the way to invoke the m-service: parameters that are needed, their type, etc. It uses a subset of WSDL language that contains `types`, `message`, and `portType` elements [18]. The remaining elements (`binding`, `port`, and `service`) are not used since they are specific to Internet-based invocation of Web service(e.g., HTTP and MIME). The *data channel* contains all data needed to execute each m-service within the m-service channel. For better access performance, information within the data channel is also indexed using a primary key (e.g., restaurant address).

Executing M-services – We present in Figure 2 an algorithm with the main steps that need to be performed to execute an m-service (i.e., invoking one of its operation). Mobile users generally start by looking for m-service based on their category (e.g., restaurants, hotels). They may also look for m-services based on their names. If the m-service `serviceKey` is known *a priori* by the mobile user then steps 1 and 2 in the algorithm are skipped.

Walk-through Scenario – To illustrate our approach, let us use the `restaurant_finder` m-service as an example. Consider a tourist in Washington DC equipped with a wireless device. After enjoying different attractions in the city, the tourist would like to go to a Chinese restaurant. For that purpose, she/he accesses the cached UDDI registry looking for an m-service that returns the list of local restaurants. The tourist executes a UDDI inquiry that returns m-services having "restaurants" as a category. Assume the registry returns two m-services. Since the first m-service does not search for restaurants based on their specialties (Chinese food), the tourist selects the second m-service i.e., `restaurant_finder`. The wireless device then listens to the m-service channel and uses the `servicekey` returned by the UDDI inquiry to download the `restaurant_finder`'s code and description. The `restaurant_finder`'s description specifies an operation `return_list_of_restaurants` that requires the user's street address and restaurant specialty as input. The tourist invokes that operation by giving the required input (i.e., her/his current location and "Chinese food" as a specialty). During `restaurant_finder`'s execution, the wireless device listens to the data channel and downloads all data whose address (primary data key) corresponds to the tourist's current location. The m-service execution is then resumed. The data downloaded from the data channel is filtered out. Only restaurants providing Chinese food are returned to the tourist.

III. Wireless Channel Access Methods

Two important factors are normally used to measure the performance of data access in wireless environments: *Access Time* and *Tuning Time*. Access time refers to the total time mobile clients need to wait for the request to complete. Tuning time is the actual time spent by mobile clients to actively listen to wireless channels and process requests. Processing requests

requires CPU to be active and listening to wireless channels implies that receiving devices are receiving data from wireless channels. Since most power consuming parts of a mobile unit are CPU and receiving devices [19], the tuning time of a request is usually proportional to the power consumed by a mobile unit on the request.

Applying this concept to our m-service environment, the access time and tuning time reflect two important aspects of the m-service system, the response time and energy efficiency. In the context of m-service system, we define access time and tuning time as follows:

- **Access Time:** This is the total client waiting time, starting from the issuance of a service request till its completion.
- **Tuning Time:** Tuning time is the time when CPU and/or wireless receiving devices are active, which means mobile clients are either actively processing requests and/or retrieving data from wireless channels. Tuning time includes the time to download or update the UDDI registry (if required), find and download the requested m-service, execute the m-service, and retrieve the requested data items.

Mobile clients in m-service environment are usually equipped with limited power supply, such as batteries. This requires more power efficient access to wireless channels. In [10], a few typical wireless data access techniques, namely B+ tree indexing, signature indexing and hashing, are discussed. These techniques can improve the access efficiency to wireless channels. All of these techniques are based on the idea of using extra information in wireless channels to help locate the requested data. As a result, mobile clients do not need to actively listen to wireless channels all the time in order to preserve power consumed by receiving devices. We define bucket as the logical broadcast unit in wireless channels. For different access methods, buckets may have fixed or varied sizes. There are also different types of buckets based on their contents. For example, we refer to buckets containing only broadcast data as data buckets. In B+ tree indexing, buckets containing indices are called index buckets, and signature buckets in signature indexing contain only signatures.

In this section, we will discuss how these techniques can be applied to our m-service infrastructure. For each data access techniques, we consider the following three scenarios: (1) access to m-service channel broadcasting fixed-size m-services; (2) access to m-service channel broadcasting varied-size m-services; and (3) access to data channel We make the reasonable assumption that data items in the data channel have the same size.

A. B+ Tree Indexing

The use of B+ tree indexing in wireless environments is very similar to that of traditional disk based environments. Indices are organized in B+ tree structure to accelerate search processes. An offset value is stored in each index node pointing at corresponding data item or lower level index node. However, there are some differences that introduce new challenges to wireless environments. For example, in disk based environments, offset value is the location of the data item on disk, whereas in wireless environments, offset value is the arrival time of the bucket containing the data item. Moreover, indices and data in wireless environments are organized in one-dimensional mode in broadcast channel. Missing the bucket containing index of the requested data item may cause the client to wait until the next broadcast cycle to find it again. Two indexing techniques, *(1,m) indexing* and *distributed indexing*, which are both based on B+ tree data structure, are presented in [15]. We compared these two techniques in [10] and found that they exhibit similar performance except that (1,m) indexing has slightly better tuning time and distributed indexing has better access time. In this paper, we only focus on (1,m) indexing.

Generally, we refer to the information being broadcast in wireless channel as data items. In the context of m-service system, the broadcast data items can be either m-services or database records. In broadcast channel using (1, m) indexing as the access method, every broadcast data is indexed on its key attribute. For m-services, the key attribute is the service key of each m-service and for database it is the primary key of each database record. Indices are organized in B+ tree structure, which is referred to as *index tree*.

Each index node has a number of pointers pointing at its child nodes. The pointers of the bottom level indices point at the actual data nodes. To find a specific broadcast data item, the search follows a top-down approach. The top level index node is searched first to determine which child node contains the data item. Then the same process will be performed on that node. This procedure continues till it finally reaches the data item at the bottom. The sequence of the index nodes traversed is called the *index path* of the data item. A node in the index tree represents an index bucket in the broadcast channel. A broadcast data item is represented by a data bucket. In the traditional disk-based systems, index and data are usually stored in different locations. The index tree is searched first to obtain the exact location of the requested data item. This process often requires frequent shifts between index nodes or between index and data nodes. As data in a wireless channel is one-dimensional, this kind of shift is difficult to achieve. Therefore, in (1, m) indexing, data and index are interleaved in the broadcast channel. The broadcast data is partitioned into several data segments. The index tree precedes each data segment in the broadcast. Users traverse the index tree first to obtain the time offset of the requested data item. Then they switch to doze mode until the data item arrives. Figure 3 illustrates how index and data are organized in the broadcast channel.

In (1,m) indexing, the whole index tree precedes each data segment in the broadcast. Each index bucket is broadcast a number of times, *m*, equal to the number of data segments. Each index bucket contains pointers to the buckets containing its child nodes. When a mobile client tunes into the broadcast channel, it is redirected to the beginning of the next index segment. From there, it follows the index path to find the requested data item. Between any two consecutive probes, the mobile client can go to doze mode to save power.

In the original (1,m) indexing method, all broadcast buckets

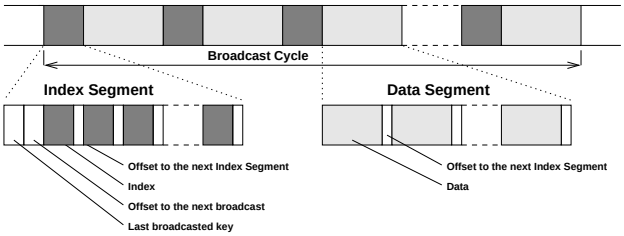

Fig. 3. Index and data organization of (1,m) indexing

are of the same size. The bucket size is determined by the length of a broadcast data item (in this case, a m-service) plus an offset value. Each bucket may contain a few index entries depending on the size of data items and their primary keys. However, in the m-service channel, where m-services are much larger than the service keys, a bucket with the size equivalent to a m-service may contain many index entries up to the whole index tree. This obviously loses the flexibility of (1,m) indexing, because every time a mobile client needs to read index information, it has to download many unnecessary index entries or maybe even the whole tree. To preserve the flexibility of (1,m) indexing, for m-service channel, we use an improved (1,m) indexing method, in which index buckets have smaller size than m-services. The structure of the broadcast channel will still be the same as shown in Figure 3, except that the index buckets are smaller. Since the pointers stored in each index bucket are the absolute time offset to the child index nodes or data buckets, the size of the data buckets does not have any impact on the wireless channel structure.

B. Signature Indexing

A signature is essentially an abstraction of the information stored in a record. It is generated by a specific signature function. By examining a data item's signature, one can tell if the record possibly has the matching information. Since the size of a signature is much smaller than that of the data item itself, it is considerably more power efficient to examine signatures first instead of simply searching through all data items. In [20], three signature indexing schemes are proposed: simple signature, integrated signature, and multi-level signature. In this section, we introduce the basic signature indexing technique these schemes are based on and discuss how this technique can be applied to m-service system.

When used with databases, the signatures are generated based on all attributes of data records. A signature is formed by hashing each field of a record into a random bit string and then superimposing together all the bit strings into a record signature. The number of collisions depends on how perfect the hashing function is and how many attributes a data record has. Collisions in signature indexing occur when two or more data records have the same signature. Usually the more attributes each data record has, the more likely collisions will occur. Such collisions would translate into false drops, where clients download wrong data records but with matching signatures.

With basic signature indexing technique, signatures are broadcast together with data records. The broadcast channel consists of signature buckets and data buckets. Each broadcast of a data bucket is preceded by a broadcast of the signature bucket, which contains the signature of the data record. All signature buckets have equal length. Mobile clients must sift through each broadcast bucket until the required information is found. The data organization of signature indexing is illustrated in Figure 4.

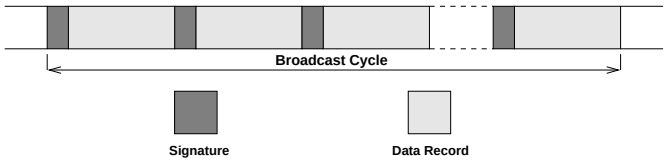

Fig. 4. Data organization of signature indexing

We now discuss how the signature indexing technique can be used in our m-service infrastructure. In m-service channel, the broadcast data items are the m-services. Preceding each m-service, the corresponding service key is broadcast as the signature of that m-service. Mobile clients will read the service key first before downloading the whole m-service. Since the service key is unique, there will be no false drops if service key is used as signature directly. If the m-service size is fixed, the m-service channel structure will be the same as what is shown in Figure 4. When the sizes of m-services are different, an extra field indicating the size of the followed m-service bucket is needed in each signature bucket in the m-service channel. In other words, each signature bucket shown in Figure 4 will consist of two parts: the signature and the size of the m-service that follows. The size of the m-service here is expressed as the time offset to the next signature bucket. In the data channel, since we assume that all database records are equal in length, the data channel structure will be the same as in Figure 4.

C. Hashing

Hashing techniques store hashing parameters in data buckets without requiring separate index buckets or segments [16]. Each data bucket consists of two parts: *Control part* and *Data part*. The *Data part* contains actual data records and the *Control part* is used to guide clients to the right data bucket. The data organization of the broadcast channel using simple hashing is illustrated in Figure 5.

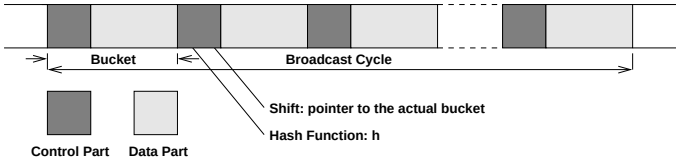

Fig. 5. Index and data organization of simple hashing

The control part of each data bucket consists of a *hashing function* and a *shift value*. The hashing function maps the key value of the broadcast data into a hashing value. Each bucket has a hashing value *H* assigned to it. In the event of a collision, the colliding data is inserted right after the

bucket which has the same hashing value. This will cause the rest of the buckets to shift, resulting in broadcast data being "out-of-place". The shift value in each bucket is used to find the right position of the corresponding data. It points to the first bucket containing the data with the right hashing value. Assume the initial allocated number of buckets is N_a. Because of collisions, the resulting length of the broadcast cycle after inserting all the broadcast data will be greater than N_a. The control part of each of the first N_a buckets contains a shift value (offset to the bucket containing the actual data with the right hashing value), and the control part of each of the remaining data buckets contains an offset to the beginning of the next broadcast.

In m-service channel with fixed-size m-services and data channel, the broadcast channel will have the same structure as shown in Figure 5 because all broadcast items are equal in size. For m-service channel with varied-size m-services, the same technique cannot be directly applied due to the fact that all m-services have different sizes. We propose a improved hashing technique that can deal with varied-size broadcast items. In the improved hashing technique, we use a small size broadcast bucket. As a result, every m-service may take one or more buckets to store. The broadcast channel is constructed as follows:

- Allocate N_m empty buckets, where N_m is the number of m-services.
- Generate a hash code for each m-service using the service key.
- Place each m-service in the allocated bucket based on its hash code.
- Calculate the number of buckets used to store this m-service and place the value in the first bucket of this m-service.
- For each m-service, consider the following scenarios:
  - If the m-service size is larger than the bucket size, keep creating new buckets until the m-service can fit in and shift all the rest of buckets forward.
  - When there is a hash code conflict, create one or more buckets to fit the m-service and shift the rest of buckets forward.
  - Change the shift value accordingly whenever a bucket is shifted forward.

There are two cases that can cause a bucket to shift forward: (1) when there is a hashing conflict, this is the same as in fixed size case; (2) when a m-service is larger than the bucket size. As a result, in the broadcast channel, all buckets will still be equal in size, except that some m-services may occupy several buckets. However, the structure of the broadcast channel is still the same as that in fixed size m-service channel. Therefore, the way of accessing m-service channel with varied-size m-services will be the same too.

Upon tuning into the broadcast channel, a mobile client will download the first complete bucket it comes across. Then the mobile client calculates the hash code of the request service key or primary key of the requested data item and compare the calculated hash code with the hash code stored in the current bucket to find the time offset to the right bucket (*hashing position*), which means the bucket with the matching hash code. The mobile client then goes to doze mode and wakes up when the right bucket arrives. If the shift value in that bucket is not empty, it means the actual bucket has been shifted because of collisions or varied-size m-services. The mobile client then goes to the bucket indicated by the shift value (*shift position*) to retrieve the requested service or data item.

IV. Analytical Model

In this section, we present the analytical model for each access method. The total access time for each service request will be the sum of the time to obtain UDDI registry, download and execute the selected m-service, and retrieve the data items. The tuning time will be the time spent to actively listen to the m-service and data channels plus the time to execute the m-service. Table I defines the parameters and symbols used in this section. The access time and tuning time can be expressed as follows:

$$A_t = U_t + C_{tm} + A_{tm} + M_t + C_{td} + A_{td}$$
$$T_t = U_t + T_{tm} + M_t + T_{td}$$

Variables	
N_m	Number of registered m-services
S_m	M-service program size
S_{mk}	Size of m-service key
N_o	Number of operations per service
N_d	Number of data items in database
S_{dk}	Key size of data items
S_d	Data item size
S_b	Logical broadcast unit (bucket) size
B_c	Broadcast cycle - the length of all contents in the broadcast channel
B_m	M-service wireless channel bandwidth
B_d	Database wireless channel bandwidth
Performance measurement parameters	
A_t	Total access time
T_t	Total tuning time
U_t	Time to obtain UDDI information
C_{tm}	Time to switch to m-service channel
A_{tm}	Access time to obtain m-service
T_{tm}	Tuning time to obtain m-service
M_t	Time to execute m-service
C_{td}	Time to switch to data channel
A_{td}	Access time to retrieve data items
T_{td}	Tuning time to retrieve data items
B_t	Broadcast cycle time - time to scan the whole broadcast channel

TABLE I
SYMBOLS AND PARAMETERS

Our focus of this paper is on access methods for m-services and data channels. To simplify the analysis, we assume that the time to obtain UDDI registry is a fixed value. It is also reasonable to assume that the switch time between two channels is a fixed value. Thus, in the above formulas, U_t, C_{tm}, M_t, and C_{td} are constants. Now what account for the access efficiency of each service request are the access time and tuning time of m-service and data channels. In the rest of this section, we will show how to derive access and tuning

times for accessing m-service channel and data channel with different access methods.

A. Accessing m-service channel

In this section, we investigate three different access methods that can be used to make m-service channel access more efficient, namely, *signature indexing*, *hashing*, and *B+ tree indexing*. We consider two scenarios for each method: (1) m-service channel contains fixed-size m-services; (2) m-service channel contains varied-size m-services.

1) No data access method: As a comparison benchmark, we present the performance of accessing m-service channel without access methods. Mobile clients need to traverse through the whole broadcast channel until the requested m-service is found. Therefore, the expected access and tuning times will not be dependent on whether the m-service size is fixed or not. When a mobile client tunes into a broadcast channel, it may hit any position of a broadcast bucket. The mobile client has to stay active until the first complete bucket is retrieved so as to acquire enough access information. We define the time spent for the first complete broadcast bucket to arrive as the *initial wait time* (F_t). The average access time and tuning time are both half of the whole broadcast cycle time, plus the initial wait time:

$$\begin{aligned} A_{tm} &= T_{tm} = \frac{1}{2} \times B_t + F_t \\ &= \frac{1}{2} \times (N_m + 1) \times \frac{S_m}{B_m} \end{aligned}$$

On average, the initial wait time is equal to the time to retrieve half of a bucket, which is $\frac{1}{2} \times \frac{S_m}{B_m}$. The initial wait time will be part of the access time and tuning time of every access method, but the calculation may vary.

2) M-service channel with fixed-size m-services:

Signature indexing: In this method, mobile clients always examine the preceding service key before downloading a m-service. Since the service key of each m-service is unique, there will be no false drop. The average access time is half of the broadcast cycle time plus the initial wait time. The tuning time consists of the initial wait time, time to retrieve $\frac{N_m}{2}$ signature buckets and download the requested m-service. The average initial wait time here is half of the time to scan a signature bucket and a m-service bucket, which is as $\frac{1}{2} \times (S_m + S_{mk})/B_m$. There are N_m signature buckets and N_m m-service buckets in a broadcast cycle. Thus, the broadcast cycle time is $N_m \times (S_m + S_{mk})/B_m$. Based on the above analysis, we have:

$$\begin{aligned} A_{tm} &= \frac{1}{2} \times \frac{(S_m + S_{mk})}{B_m} + \frac{1}{2} \times N_m \times \frac{(S_m + S_{mk})}{B_m} \\ &= \frac{1}{2} \times (N_m + 1) \times \frac{(S_m + S_{mk})}{B_m} \\ T_{tm} &= \frac{1}{2} \times \frac{(S_m + S_{mk})}{B_m} + \frac{1}{2} \times N_m \times \frac{S_{mk}}{B_m} + \frac{S_m}{B_m} \\ &= (1\frac{1}{2} \times S_m + \frac{1}{2} \times (N_m + 1) \times S_{mk})/B_m \end{aligned}$$

Hashing: In this method, each service key is hashed to an integer value. The hashing function precedes each m-service to help locate the requested service key. The hash value of each service key also precedes the corresponding m-service. In cases of collisions, buckets in broadcast channel may be out of place. Therefore, an offset value is required at the beginning of each bucket to indicate the right position of the m-service with the correct hashing value. Let S_h be the hashing function size, S_{hk} be the hash value size, and S_{of} be the offset value size. Thus, the size of each bucket S_b is $S_m + S_h + S_{hk} + S_{of}$.

The access time of the hashing method consists of the initial wait time (F_t), time to reach the *hashing position* (H_t), time to reach the *shift position* (S_t), time to retrieve colliding buckets (C_t), and time to download the required bucket (D_t). D_t is the time to read one complete bucket, which is $\frac{S_b}{B_m}$. Let N_c be the number of colliding buckets, if the collisions are uniformly distributed among all m-services, the average number of shifts for each bucket is thus $\frac{N_c}{2}$. Therefore, we have $S_t = \frac{N_c}{2} \times \frac{S_b}{B_m}$. Furthermore, the average number of colliding buckets for each hashing value is $\frac{N_c}{N_m}$. Thus, we have $C_t = \frac{N_c}{N_m} \times \frac{S_b}{B_m}$. There is more involved in the calculation of H_t. Assume the number of initially allocated buckets is the number of m-services N_m and the number of colliding buckets is N_c. The resulting total number of buckets in the broadcast cycle is $N = N_m + N_c$. We calculate H_t based on the position of the first arriving bucket and whether the requested information has been broadcast or not. Assuming that the position of the first arriving bucket is *n*, H_t consists of the following three parts:

$$H_{ti} = \begin{cases} H_{t1} & (n > N_m) \\ H_{t2} & (n \leq N_m \text{ and } req\_item\_broadcast = False) \\ H_{t3} & (n \leq N_m \text{ and } req\_item\_broadcast = True) \end{cases}$$

The *req_item_broadcast* above designates if the requested information has already been broadcast in the current broadcast cycle. Each part of H_t is derived as follows:

$$\begin{aligned} H_{t1} &= \frac{N_c}{N} \times (\frac{1}{2} \times (N_c + N_m)) \times \frac{S_b}{B_m} = \frac{1}{2} \times N_c \times \frac{S_b}{B_m} \\ H_{t2} &= \frac{1}{2} \times \frac{N_m}{N} \times \frac{N_m}{3} \times \frac{S_b}{B_m} \\ H_{t3} &= \frac{1}{2} \times \frac{N_m}{N} \times (\frac{N_m}{3} + N_c + \frac{N_m}{3}) \times \frac{S_b}{B_m} \end{aligned}$$

The first part of each formula above is the probability the scenario will happen. As a result, we have $H_t = H_{t1} + H_{t2} + H_{t3}$. Based on the above discussion, the access time can be derived as:

$$\begin{aligned} A_{tm} &= F_t + H_t + S_t + C_t + D_t \\ &= (N_c + \frac{1}{2} \times N_m + \frac{N_c}{N_m} + 1\frac{1}{2}) \\ &\quad \times \frac{(S_m + S_h + S_{hk} + S_{of})}{B_m} \end{aligned}$$

The tuning time consists of the initial wait time, the time to read the first bucket to obtain the hashing position (read one bucket), time to obtain the shift position (read one bucket), and

time to retrieve the colliding buckets (C_t), and time to download the required bucket (read one bucket). The probability of collision is $\frac{N_c}{N_m}$. Thus, we have $C_t = \frac{N_c}{N_m} \times S_b/B_m$. For those requests that tune in at the time which the requested bucket has already been broadcast, one extra bucket read is needed to start from the beginning of the next broadcast cycle. The probability of this scenario occurrence is $(N_c + \frac{1}{2} \times N_m)/(N_c + N_m)$. As a result, the expected tuning time is:

$$T_{tm} = (\frac{N_c + \frac{1}{2} \times N_m}{N_c + N_m} + \frac{N_c}{N_m} + 3\frac{1}{2}) \times \frac{S_b}{B_m}$$

where $S_b = S_m + S_h + S_{hk} + S_{of}$.

B+ tree indexing: Let n be the number of index entries can be stored in a bucket, intuitively $n = \lfloor \frac{S_m}{S_{mk}} \rfloor$. Let k be the number of levels of the index tree. It is also intuitive that $k = \lceil \log_n(N_m) \rceil$. The access time consists of three parts: the initial wait time, *initial index probe time*, and *broadcast wait time*. Let S_{ib} be the size of index bucket and S_{mb} be the size of m-service buckets. We have the following derivations:

initial wait (F_t): The calculation of the initial wait is different from that of signature indexing because the number of index buckets is not the same as the number of m-service buckets. When a client tunes into the broadcast channel, it may hit an index or a data bucket arbitrarily. Let N_i be the number of index buckets in a broadcast cycle, the probability of hitting a index bucket is $\frac{m \times N_i \times S_{ib}}{B_c}$ and m-service bucket is $\frac{N_m \times S_{mb}}{B_c}$. Therefore, the initial wait is:

$$F_t = \frac{1}{2} \times (\frac{m \times N_i \times S_{ib}}{B_c} \times S_{ib} + \frac{N_m \times S_{mb}}{B_c} \times S_{mb})/B_m$$

initial index probe (P_t): This part is the time to reach the first index segment. It can be expressed as the average time to reach the next index segment, which is calculated as the sum of the average length of index segments and data segments. With an *n*-ary index tree of *k* levels, the number of index buckets (N_i) is:

$$N_i = 1 + n + ... + n^{k-1} = \frac{n^k - 1}{n - 1}$$

where $k = \lceil \log_{\lfloor \frac{S_{ib}}{S_{mk}} \rfloor}(N_m) \rceil$. The average number of data buckets in a data segment is $\frac{N_m}{m}$. Therefore, the initial index probe is calculated as:

$$P_t = \frac{1}{2} \times (\frac{n^k - 1}{n - 1} \times S_{ib} + \frac{N_m}{m} \times S_{mb})/B_m$$

broadcast wait (W_t): This is the time from reaching the first indexing segment to finding the requested m-service. It is approximately half of the whole broadcast cycle time B_c, which is $(m \times N_i \times S_{ib} + N_m \times S_{mb})/B_m$. Thus, the total access time is:

$$A_{tm} = F_t + P_t + W_t$$

The tuning time is easier to calculate than access time, because during most of the probes clients are in doze mode. It includes the initial wait F_t, the time to read the first bucket to find the first index segment (read one index or data bucket, which is $2 \times F_t$), the time to traverse the index tree (read k buckets), and the time to download the m-service (read one bucket). Thus, the tuning time can be derived as:

$$\begin{aligned} T_{tm} &= F_t + 2 \times F_t + (k \times S_{ib} + S_{mb})/B_m \\ &= 3 \times F_t + \frac{k \times S_{ib} + S_{mb}}{B_m} \end{aligned}$$

3) M-service channel with varied-size m-services:

Signature indexing: Since the m-services have different sizes, in front of each m-service we also need the size of the m-service to direct the mobile clients to the next m-service. Assume the size of the variable containing m-service size is S_{ms}, we have

$$\begin{aligned} A_{tm} &= \frac{1}{2} \times (N_m + 1) \times \frac{S_m + S_{mk} + S_{ms}}{B_m} \\ T_{tm} &= \frac{\frac{1}{2} \times (S_m + S_{mk} + S_{ms}) + \frac{1}{2} \times N_m \times (S_{mk} + S_{ms}) + S_m}{B_m} \\ &= \frac{1\frac{1}{2} \times S_m + \frac{1}{2} \times (N_m + 1) \times (S_{mk} + S_{ms})}{B_m} \end{aligned}$$

Hashing: The analysis process of the analytical model for hashing in case of varied m-service size will be the same as that of fixed m-service size. All m-services will still be hashed into different positions in the broadcast channel by using a hash function. However, each m-service may need one or more buckets to store it. Furthermore, now we have two reasons that could cause m-services to be out of place: the large size of m-services and the collisions. Let N_{co} be the sum of conflict and offset items, $\overline{n}$ be the average number of buckets per service. Following the same analysis process in Section IV-A.2, we can derive the access time and tuning time as follows:

$$\begin{aligned} F_t &= \frac{1}{2} \times \frac{S_b}{B_m} \\ H_{t1} &= \frac{N_{co}}{N} \times (\frac{1}{2} \times (N_{co} + N_m)) \times \frac{S_b}{B_m} \\ H_{t2} &= \frac{1}{2} \times \frac{N_m}{N} \times \frac{N_m}{3} \times \frac{S_b}{B_m} \\ H_{t3} &= \frac{1}{2} \times \frac{N_m}{N} \times (\frac{N_m}{3} + N_c + \frac{N_m}{3}) \times \frac{S_b}{B_m} \\ S_t &= \frac{N_{co}}{2} \times \frac{S_b}{B_m} \\ C_t &= \frac{N_c}{N_m} \times \frac{S_b}{B_m} \\ D_t &= \overline{n} \times \frac{S_b}{B_m} \\ A_{tm} &= F_t + H_t + S_t + C_t + D_t \\ &= (N_{co} + \frac{1}{2}\frac{N_m^2}{N} + \frac{1}{2}\frac{N_m \times N_c}{N} + \frac{N_c}{N}_m + \overline{n} + \frac{1}{2}) \\ &\quad \times S_b/B_m \\ T_{tm} &= (\frac{N_{co} + \frac{1}{2} \times N_m}{N_c + N_m} + (\frac{N_c}{N_m} + 1) \times \overline{n} + 2\frac{1}{2}) \times \frac{S_b}{B_m} \end{aligned}$$

B+ tree indexing: The analytical model of B+ tree indexing for varied-size m-service channel is the same as the one for fixed-size m-service channel. The time offset values contained

in each bucket are absolute values. The differences in the m-services sizes only result in the differences in the values of those time offsets. The expressions of the access time and tuning time are still the same as those presented in Section IV-A.2.

B. Accessing data channel

In this section, we present the analytical model for each access method being applied to the data channel when each request acquires one or more data items. Let N_o be the number of operations per mobile service and N_{do} be the number of data items requested by an operation. The total number of data items requested per m-service is thus $N_o \times N_{do}$. Without losing the generality, to simplify the analysis, we assume that every operation only requests for one data item. Therefore, for each m-service, N_o data items will be requested.

No data access method: In this case, no access method is used to access data in the data channel. Mobile clients need to traverse through the whole broadcast channel until all the requested N_o data items are found. If all data items are uniformly distributed in the data channel, the average traversing distance to reach the last requested data item will be $\frac{N_o}{N_o+1}$ of the whole broadcast cycle time plus the initial wait time. Thus, we have

$$\begin{aligned} A_{td} &= T_{td} = \frac{N_o}{N_o+1} \times B_t + F_t \\ &= (\frac{N_o}{N_o+1} \times N_d + \frac{1}{2}) \times \frac{S_d}{B_d} \end{aligned}$$

Signature indexing: Mobile clients always examine the primary key first before downloading the whole data item. To find all requested data items, mobile clients need to finish listening to $\frac{N_o}{N_o+1}$ of the whole broadcast cycle. The access time and tuning time can be derived as follows:

$$\begin{aligned} A_{td} &= (\frac{N_o}{N_o+1} \times N_d + \frac{1}{2}) \times \frac{S_d + S_{dk}}{B_d} \\ T_{td} &= \frac{\frac{1}{2} \times (S_d + S_{dk}) + \frac{N_o}{N_o+1} \times N_d \times S_d k + N_o \times S_d}{B_d} \end{aligned}$$

Hashing: The access time and tuning time of hashing for data channel can be derived based on the analysis in Section IV-A.2. The only difference is that for data channel, N_o data items are acquired for each request. When tuning into the data channel, a mobile client downloads the first complete bucket to obtain the hashing function and the hash code of the current bucket. Then the mobile client calculates the hash code for each request key and compares it with the hash code stored in the current bucket. From the comparisons, the distance to the hashing position of each data item can be derived. The mobile client then saves these distance values in a distance list in the order of the distances and goes to them one by one following the distance list. After reaching each hashing position, the mobile client follows the same procedure as described in Section IV-A.2 to find the requested data item. At each step, the distance list is updated with any new distance value obtained. Since all these steps are done sequentially, the total access time to retrieve all N_o data items will be the access time spent to retrieve the farthest data item from the point the mobile client tunes in. The same applies to the tuning time. With this in mind, our analysis can be simplified as requesting only one data item given that the item is the farthest one of N_o uniformly distributed data items from the point a mobile client tunes in.

Based in the analysis in Section IV-A.2, the access time still consists of: F_t, H_t, S_t, and C_t, plus the time to download the requested data items (read N_o buckets). If the position of the first arriving bucket is after N_d, the average time to get the farthest data item will be $\frac{N_o}{N_o+1} \times (N_c + N_d) \times \frac{S_b}{B_d}$; if the first arriving bucket is in N_d, and any of the requested items is not broadcast yet, the average time to get the farthest data item will be $\frac{N_o}{N_o+2} \times N_d \times \frac{S_b}{B_d}$; if the first arriving bucket is in N_d, and one of the requested items is not broadcast yet, the average time to get the farthest data item will be $(\frac{N_o+1}{N_o+2} \times N_d + N_c) \times \frac{S_b}{B_d}$. With the special consideration of the probability, each of this time can be derived as follows:

$$\begin{aligned} H_{t1} &= \frac{N_c}{N} \times (\frac{N_o}{N_o+1} \times (N_c + N_d)) = \frac{N_o}{N_o+1} \times N_c \times \frac{S_b}{B_d} \\ H_{t2} &= \frac{1}{N_o+1} \times \frac{N_d}{N} \times \frac{N_o}{N_o+2} \times N_d \times \frac{S_b}{B_d} \\ H_{t3} &= \frac{N_o}{N_o+1} \times \frac{N_d}{N} \times (\frac{N_o+1}{N_o+2} \times N_d + N_c) \times \frac{S_b}{B_d} \\ S_t &= \frac{1}{2} \times N_c \times \frac{S_b}{B_d} \\ C_t &= \frac{N_o}{2} \times \frac{N_c}{N_d} \times \frac{S_b}{B_d} \\ D_t &= N_o \times \frac{S_b}{B_d} \\ F_t &= \frac{1}{2} \times \frac{S_b}{B_d} \end{aligned}$$

The bucket size S_b above is equal to $S_d + S_h + S_{hk} + S_{of}$. The access time can be derived from the above formulas as follows:

$$A_{tm} = H_t + S_t + C_t + D_t + F_t$$

The tuning time consists of the initial wait time, the time to read the first bucket to obtain the hashing positions for all requested data items (read one bucket), time to obtain the shift position (read N_o bucket), and time to retrieve the colliding buckets ($C_t \times N_o$), and time to download the required bucket (read N_o buckets). The probability of collision is $\frac{N_c}{N_d}$. Thus, we have $C_t = \frac{N_c}{N_d} \times \frac{S_b}{B_d}$. For those requests that tune in at the time when the farthest requested bucket has already been broadcast, one extra bucket read is needed to start from the beginning of the next broadcast cycle. The probability of this scenario occurrence is $((N_c + \frac{N_o}{N_o+1} \times N_d)/(N_c + N_d)) \times \frac{S_b}{B_d}$. As a result, considering the initial wait time, the expected tuning time is:

$$\begin{aligned} T_{tm} &= (1\frac{1}{2} + N_o + \frac{N_c}{N_d} \times N_o + N_o + \frac{N_c + \frac{N_o}{N_o+1} \times N_d}{N_c + N_d}) \\ &\quad \times \frac{S_b}{B_d} \end{aligned}$$

B+ tree indexing: When requesting for multiple data items (N_o), we need a local list to save a sequence of time offset values to the buckets to be visited next. Figure 6 shows a typical full index tree consisting of 81 data items, with each index containing three child nodes. Using Figure 6 as an example, we illustrate how multiple data items are retrieved by a request.

Assume we are requesting for data items 12 and 66. The following steps are taken to retrieve these two data items:

- Mobile client tunes into data channel.
- Goes to doze mode and wakes up at the beginning of the closest index segment.
- Compares the keys of the requested data items (12 and 66) with the index entries in the root node/bucket. And saves the time offset values to a1 and a3 in a local list with a1 in front because a1 is closer.
- Goes to bucket containing node a1 and compares the keys again. Then saves b2 in the list with the new sequence a3, b2 because a3 is closer. Offset value to a1 is removed because it is already visited.
- Goes to bucket containing a3 and compares the keys. Removes a3 and saves b8 in the list with the sequence b2, b8.
- Goes to b2, compares the keys, and saves c5 after b8 in the list.
- Goes to b8, compares the keys, and saves c23 after c5 in the list.
- Goes to bucket containing data item 12 and download the data item.
- Goes to bucket containing data item 66 and download the data item.

Following the above procedure, the access time is equal to the time spent to retrieve the data item pointed by the index entry stored in the right most index bucket in the index tree. Based on the analysis in Section IV-A.2, the access time consists of the initial wait time (F_t), the initial index probe (P_t) and the broadcast wait (W_t). Let S_{db} be the size of data buckets. Since the index and data buckets have same size in data channel, the initial wait time can be simplified as follows:

$$F_t = \frac{1}{2} \times \frac{S_{db}}{B_d}$$

The derivation of initial index probe is unchanged, which is:

$$P_t = \frac{1}{2} \times (\frac{n^k - 1}{n-1} + \frac{N_d}{m}) \times \frac{S_{db}}{B_d}$$

The broadcast wait will be approximately $\frac{N_o}{N_o+1}$ of the whole broadcast cycle, which is $\frac{N_o}{N_o+1} \times N_d \times \frac{S_{db}}{B_d}$. Thus, the total access time is:

$$\begin{aligned} A_{td} &= F_t + P_t + W_t \\ &= (\frac{1}{2} + \frac{n^k - 1}{2(n-1)} + \frac{N_d}{2m} + \frac{N_o \times N_d}{N_o + 1}) \times \frac{S_{db}}{B_d} \end{aligned}$$

where $k = \lceil \log_{\lfloor \frac{S_d}{S_{dk}} \rfloor}(N_d) \rceil$.

The tuning time includes the initial wait time F_t, the time to read the first bucket to find the first index segment (read one bucket), the time T_i to traverse the index tree for N_o data items, and the time T_d to download all N_o data items, which is $N_o \times \frac{S_{db}}{B_d}$. It is hard to derive the exact value of T_i because it depends on the distribution of the N_o requested data items in the index tree. Instead, we derive the average value of T_i. The minimum value of T_i is achieved when all N_o data items are siblings in the index tree, in which K index bucket probes are required. The value of T_i reaches maximum when all N_o requested data items are distributed uniformly in the index tree. In this case we need $\log_{\lceil \frac{n}{2} \rceil}(N_m)$ probes for each data item. Therefore, the average T_i is:

$$T_i(avg) = \frac{K + N_o \times \log_{\lceil \frac{n}{2} \rceil}(N_o)}{2} \times \frac{S_{db}}{B_d}$$

As a result, the tuning time is:

$$T_{td} = F_t + P_t + T_i(avg) + T_d$$

V. Simulation and Experiments

In this section, we present our experiments to evaluate the performance of the proposed methods. The experiments are performed using an adaptive testbed [10] developed to simulate wireless data access. We further enhanced the testbed to simulate the proposed m-service environment and to be capable of supporting fixed-size and varied-size m-service channel. Since the bandwidth of wireless channels may vary in the real world, all the analytical and simulation results in this paper are presented in the form of the length of the broadcast data that has been traversed, instead of the actual time spent. Another reason that we do not use the actual time as the performance measurement is that there are several factors that may affect time measurement during simulation, such as CPU speed, network delay, CPU workload, etc. The access time for each method is represented by the length of all broadcast buckets passed by when requesting an m-service or data item. The tuning time of a request is calculated as the length of all buckets actively accessed (listened and downloaded) by the request.

We assume that every service request starts with requesting for an m-service, and after the execution of the mobile service, the mobile client will start listening to the data channel until the requested data item(s) are found and downloaded. The mobile client does not have to go back and forth between the mobile services channel and data channel. Based on this assumption, we consider the performance measurement of accessing mobile services and accessing data items to be two sequential processes and can be analyzed separately.

The experiments consist of three cases: (1) accessing fixed-size m-service channel; (2) accessing varied-size m-service channel; (3) accessing data channel. For each case, we present the simulation settings we used to evaluate the m-services and data access. The outcome of each experiment will be measured by access and tuning times.

Table II lists all common simulation settings used in our experiments. The *Confidence level* and *Confidence accuracy*

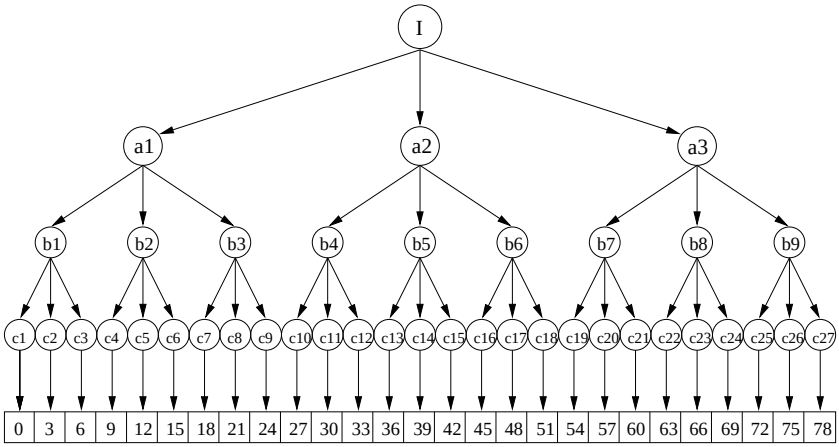

Fig. 6. A sample index tree

Mobile service key size (S_{mk})	16 bytes
Data record key size (S_{dk})	10 bytes
Data record size (S_d)	100 bytes
Number of requests	$> 10,000$
Confidence level	0.95
Confidence accuracy	0.05
Requests interval	exponential distribution
Requests generation distribution	uniform distribution

TABLE II
PARAMETERS IN THE EXPERIMENTS

	Access Method	N_m	S_m
Experiment 1	plain	*1 - 1000*	50 KB
Experiment 2	Signature indexing	*1 - 1000*	50 KB
Experiment 3	Hashing	*1 - 1000*	50 KB
Experiment 4	B+ tree indexing	*1 - 1000*	50 KB

TABLE III
ACCESSING BROADCAST CHANNEL WITH FIXED-SIZE MOBILE SERVICES

in the table are used to control the accuracy of the simulation results[1]. Users can specify the values of confidence level and accuracy before starting simulation. The simulation will not end until the expected confidence level and accuracy are achieved. By using these two parameters, we ensure that simulation results we obtained are stable. A a rule of thumb, to achieve the confidence level and accuracy of 0.95 and 0.05, it normally takes more than 10,000 requests. The interval time between any two requests follows exponential distribution. In this paper, we assume that all mobile services or data items have equal probability of being accessed, which means the requested mobile services and data items in our simulation are selected following uniform distribution. We use the standard mobile service key size, which is 16 bytes [11].

A. Performance measurement of accessing fixed-size mobile services channel

Table III lists all the experiments we conducted for accessing fixed-size mobile services channel and their simulation settings.

Figure 7 shows the simulation results for accessing fixed-size mobile services channel using plain broadcast, (1,m) indexing, signature indexing and hashing. The lines marked with (S) are simulation results. Those marked with (A) are analytical results. We observe in both figures that the simulation results match the analytical results very well.

As shown in Figure 7, signature indexing exhibits the best performance in both access time and tuning time. The reason is that the size of the mobile services is much bigger than that of the service key, which is used as the signature for each mobile service. Therefore, the overhead introduced by adding signature for each mobile service in the broadcast channel is very small compared to the whole broadcast cycle. This results in an access time close to the plain broadcast (as shown in Figure 7), which has the optimum access time performance. The tuning time of signature indexing is determined by both signature size and the number of false drops. As we mentioned in Section IV-A.2, since all service keys are unique, there is no false drop for signature indexing. So the tuning time of signature indexing consists of the reading time of a series of signatures (looking for a match) and the time to download the requested mobile service. This value is smaller than other methods because the signature size is considerably smaller than m-service size.

To help us better understand the performance trends shown in the figures, we define two terms here, *access method overhead* and *conflict overhead*. The access method overhead means the overhead introduced to the broadcast cycle to apply access methods to the broadcast channel, such as hashing values, hashing functions, signatures, and indices. And the conflict overhead designates the overhead produced by data conflicts when applying access methods to broadcast channel. Examples of conflict overhead are false drops in signature in-

[1]Given *N* sample results $Y_1, Y_2, ..., Y_N$, the *confidence accuracy* is defined as *H/Y*, where *H* is the *confidence interval half-width* and *Y* is the sample mean of the results ($Y = (Y_1 + Y_2 + ... + Y_N)/N$). The *confidence level* is defined as the probability that the absolute value of the difference between the *Y* and μ (the true mean of the sample results) is equal to or less than *H*. *H* is defined by $H = t_{\alpha/2;N-1} \times \sigma/\sqrt{N}$ where σ^2 is the sample variance given by $\sigma^2 = \Sigma_i(Y_i - Y)^2/(N-1)$ (thus σ is the standard deviation), and *t* is the standard t distribution.

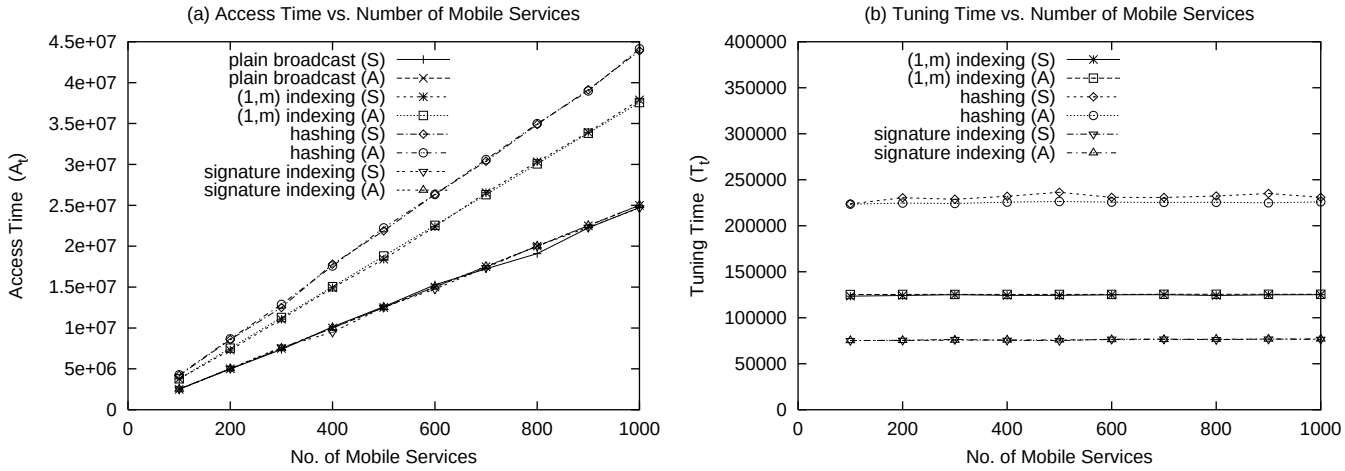

Fig. 7. Compare all access methods for fixed size m-services channel

dexing and hashing conflicts in hashing method. It is intuitive that the access method overhead is determined by the number and size of the extra information required by different access methods. For example, the access method overhead in (1,m) indexing is determined by the number of index buckets in the index tree and the number of segments. On the other hand, the conflict overhead is usually determined by the conflict rate and the size of the broadcast data, in this case, the mobile services. In the experiments covered by this section, the conflict overhead is much greater than the access method overhead because the size of m-services is a lot larger than that of service keys, which are used as signatures in signature indexing and primary key in (1,m) indexing. Since hashing is the only method that may have conflict overhead, it has the worst performance in both access and tuning times, which is also proven by the simulation results.

The performance of (1,m) indexing falls in the middle. The reasons are, on one hand, (1,m) indexing does not have conflict overhead, it thus has better performance than hashing. On the other hand, it needs to broadcast index tree *m* times in a broadcast cycle, which results in a greater access method overhead. Therefore, the resulting performance is not as good as signature indexing.

As a result, it is obvious that signature indexing is the most suitable access method for accessing broadcast channel with fixed-size mobile services.

B. Performance measurement of accessing varied-size mobile services channel

In this section, we present the simulation experiments we conducted for accessing broadcast channel that contains varied-size m-services. In these experiments, we define a range for size of m-services being broadcast and assume that the sizes follow uniform distribution in that range. Table IV shows the simulation settings. The simulation and analytical results for different methods are presented in Figure 8. Again, we observe that the simulation results match the analytical results very well.

	Access Method	N_m	S_m
Experiment 6	None	*1 - 1000*	1 - 100 KB
Experiment 7	Signature indexing	*1 - 1000*	1 - 100 KB
Experiment 8	Hashing	*1 - 1000*	1 - 100 KB
Experiment 9	B+ tree indexing	*1 - 1000*	1 - 100 KB

TABLE IV

ACCESSING BROADCAST CHANNEL WITH VARIED-SIZE MOBILE SERVICES

The access time performance shown in Figure 8(a) is very similar to that in accessing fixed-size m-service channel. The only difference is that the values are slightly larger. This is because in each method (except plain broadcast), extra information is broadcast to store the size of every m-service/bucket. (1,m) indexing and signature indexing exhibit similar performance trends in tuning time as well. However, we observe a great tuning time improvement for hashing method. This improvement is caused by the technique we used to apply hashing to varied-size m-service channel. Since hashing methods requires all buckets to the the same size, we cannot directly put every m-service in a single bucket and place the in the broadcast channel. As introduced in Section IV-A.2, we use a small size bucket as the basic broadcast unit. Every m-service may take up one or more buckets to store. By using this technique, we can still take advantage of hashing method in varied-size m-service channel. Another improvement achieved by this technique on hashing method is the improved tuning time. The conflict overhead on tuning time is now determined by the bucket size instead of the m-service size because a mobile client only needs to read the first bucket of an m-service to find out if it is the requested m-service. If not, with the help of the extra information stored in the broadcast channel indicating the size of the current m-service, the mobile client will go to doze mode and wake up when the next m-service arrives. Since the bucket size now is much smaller than the average m-service sizes, the resulting tuning time will be consequently much smaller too.

Even though the hashing method exhibits similar and sometimes even better tuning time performance than the signature

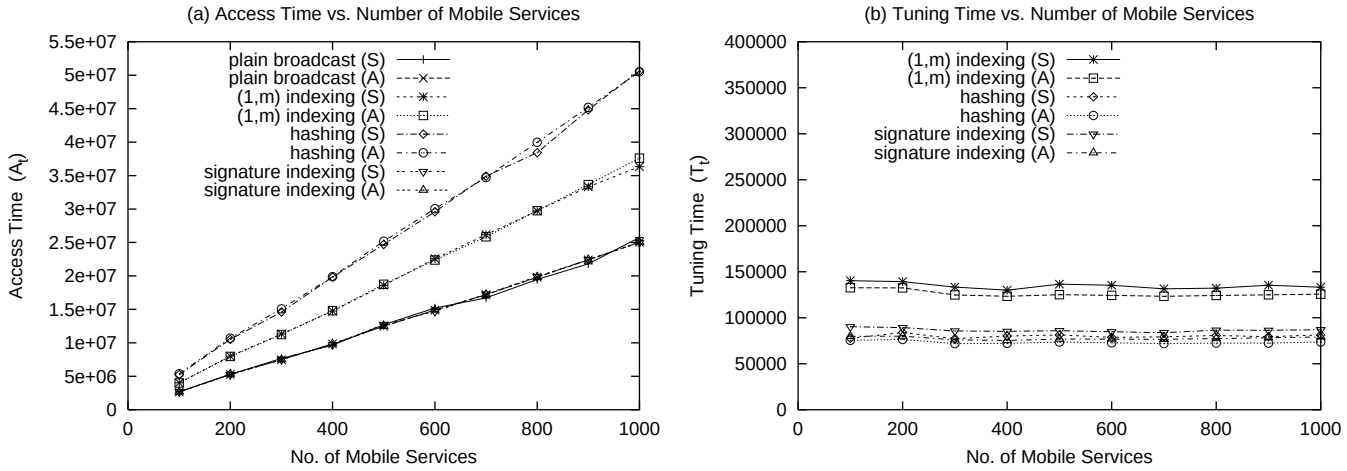

Fig. 8. Compare all access methods for varied size m-services channel

indexing, the latter is still preferred in varied-size m-service channel because of its good performance in access time. Therefore, we conclude that the signature indexing is the most suitable access method for m-services channel, either with fixed-size or varied-size m-services.

C. Performance measurement of accessing data channel

As a result of executing m-services, mobile clients will tune into data channel to request data items. In [10], we conducted extensive experimental study on performance evaluation of accessing data channel for different access methods. However, all the completed experiments assume that a mobile client only requests for one data item at a time. This is obvious not applicable in the m-service environment, where each m-service may have one or more operations, and each of such operation may request one or more data items. As discussed in Section IV-B, without losing generality, we assume that each operation requests one data item only. Therefore, each m-service requests N_o data items, where N_o is the number of operations contained in the m-service.

In this section, we present the simulation experiments for accessing data channel with multiple operations for each m-service. We consider two scenarios: (1) every m-service has fixed number of operations; and (2) every m-service has varied number of operations.

Accessing data channel with fixed number of operations: Table V lists all simulation experiments we conducted for the scenario that each m-service has 5 operations. We assume that the data channel may contain up to 100,000 data items.

	Access Method	N_o	N_d
Experiment 11	None	5	*10,000 - 100,000*
Experiment 12	Hashing	5	*10,000 - 100,000*
Experiment 13	B+ tree indexing	5	*10,000 - 100,000*
Experiment 14	Signature indexing	5	*10,000 - 100,000*

TABLE V
ACCESSING DATA CHANNEL WITH FIXED NUMBER OF OPERATIONS

We observe from Figure 9 that plain broadcast and signature still have the best access time performance which is consistent with our previous study in [10]. However, we also observe that hashing method exhibits better performance than (1,m) indexing, which is contradictory to the results of the simulation experiments we conducted before, where only one data item is acquired by a request. This phenomenon can be explained as follows: With (1,m) indexing, the whole index tree is broadcast m times in each broadcast cycle. When only one data item is requested, on average a request will traverse though about half of these trees. But when N_o data items are requested, a request will have to traverse through most of these trees depending on the value of N_o. This overhead results in a larger access time for (1,m) indexing.

Figure 9 only shows the tuning time for (1,m) indexing and hashing method because the tuning time for plain broadcast and signature indexing are too large to fit into this figure. We therefore exclude them from our analysis. As we can see from Figure 9, hashing method still has the best tuning time among all methods, which is also consistent with our previous study.

Accessing data channel with varied number of operations: We now consider the scenario that each m-service has different number of operations. This scenario is closer to the real world. We assume that the data channel contains 50,000 data items. Table VI lists all the experiment settings.

	Access Method	N_o	N_d
Experiment 15	None	*1 - 10*	50,000
Experiment 16	Hashing	*1 - 10*	50,000
Experiment 17	B+ tree indexing	*1 - 10*	50,000
Experiment 18	Signature indexing	*1 - 10*	50,000

TABLE VI
ACCESSING DATA CHANNEL WITH VARIED NUMBER OF OPERATIONS

Figure 10 shows the simulation results for accessing the data channel with varied number of operations. We observe that both access time and tuning time of every access method increase with the number of operations per m-service. It is noticeable that the increase of access time is not linear. The

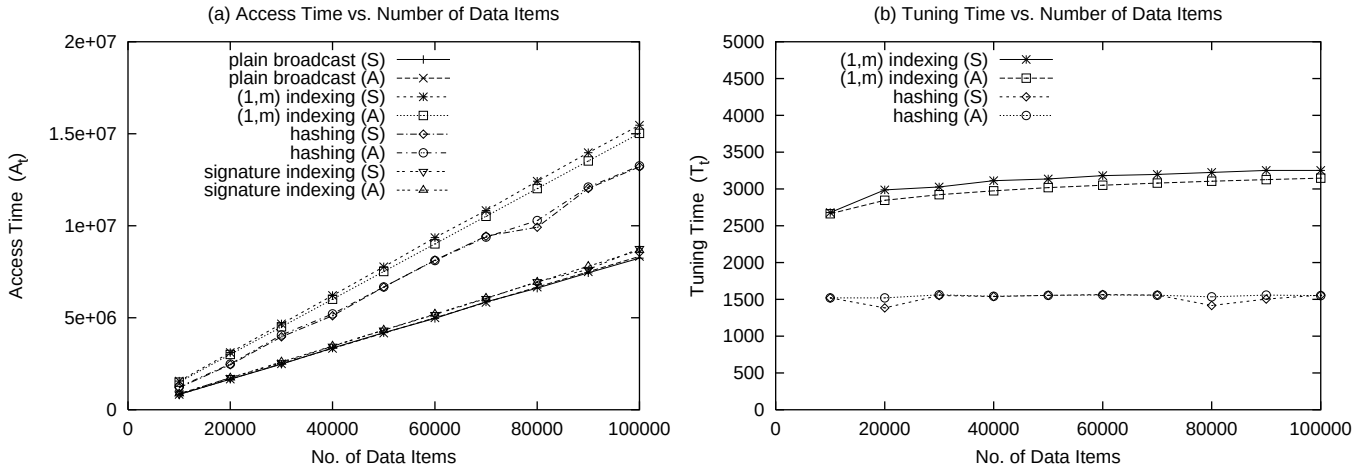

Fig. 9. Compare all access methods for data channel with fixed number of operations

increase becomes slower when the number of operations gets larger. This can be explained by using plain broadcast as an example. When requesting one data item in plain broadcast, a mobile client needs to traverse averagely half of the broadcast cycle to obtain the data item. With two data items, the mobile client needs to go through two thirds of the broadcast cycle on average, and three fourth of the broadcast cycle for three data items, four fifth for four, and so on. As the number of data items increases, the *increase* value of the traverse length decreases, which indicates the access time will increase more slowly with the number of operations. On the other hand, since the tuning time directly relates to the number of data items mobile clients download, it increases linearly with the number of operations.

Comparing the access methods, we still see the same performance trends as in the last section. Plain broadcast and signature indexing have the best access time performance. Hashing method still has the best tuning time performance and has better access time performance than (1,m) indexing.

VI. Related Work

M-commerce has attracted considerable attention during the past years. The *Wireless Access Protocol* (WAP) initiative is working on standard protocols to enable access to services from mobile hosts [21]. WAP defines a 3-tier architecture where the central component, the *gateway*, is responsible for encoding and decoding requests from wireless devices to Web servers and vice-versa. Contrary to our approach which uses the broadcast mode, WAP is mainly based on the on-demand access mode. Broadcast has been proven to be more efficient (in terms of power consumption) than the on-demand mode [15], [16]. Additionally, the use of a centralized gateway represents a single point of failure in WAP architecture. Finally, WAP does not address the issue of discovering m-services.

Mobile Data Services (MDS) is an architecture that implements SOAP protocol for wireless devices [14]. The architecture consists of two main components: an MDS portal server, which implements a SOAP server, and an MDS client which sends requests ti the MDS portal server using SOAP remote procedure calls. MDS uses the on-demand access mode for accessing m-services. Additionally, it inherits the scalability problems of SOAP protocol [1]. [22] presents an agent-based architecture to support m-services. In this architecture, storage servers play the role of intermediaries between users' and providers' agents. They store m-services to be sent to wireless devices for execution. Device-agents are defined to wrap m-services before they are sent to wireless devices. The cost of downloading m-services from storage servers to the wireless device still needs to be evaluated. Additionally, no performance study has been conducted to assess the scalability of this approach. Indexing methods have been developed to enable efficient access to wireless channels in broadcast environments [15], [16]. However, these methods are designed for mobile databases *not* for m-services. Other techniques to address location [23] and dependability [24] issues in m-commerce have also been proposed. However, these techniques do not deal with m-services organization and access.

Several companies are developing platforms to support m-services [25]. These platforms are at various development stage and operate at different levels of disclosure. IBM has released a Services Toolkit for Mobile Devices which provides tools for building Web services that can be accessed from different types of wireless devices (e.g., Palm devices, Windows CE-based Pocket PCs). Microsoft offers tools based on *.NET* technology for building Web services that run on the Pocket PC and Windows CE platforms. Nokia introduced the Nokia Multimedia Message Service which links wireless devices with Web application servers. Other industry leaders involved in Web services, such as HP, BEA Systems, and Oracle, are also providing utilities for m-services. Note that most of the exist commercial products use at-hand or on-demand modes for accessing m-services.

VII. Conclusion

In this paper, we proposed a novel infrastructure for organizing and accessing m-services in broadcast environments. Key requirements for designing such infrastructure are min-

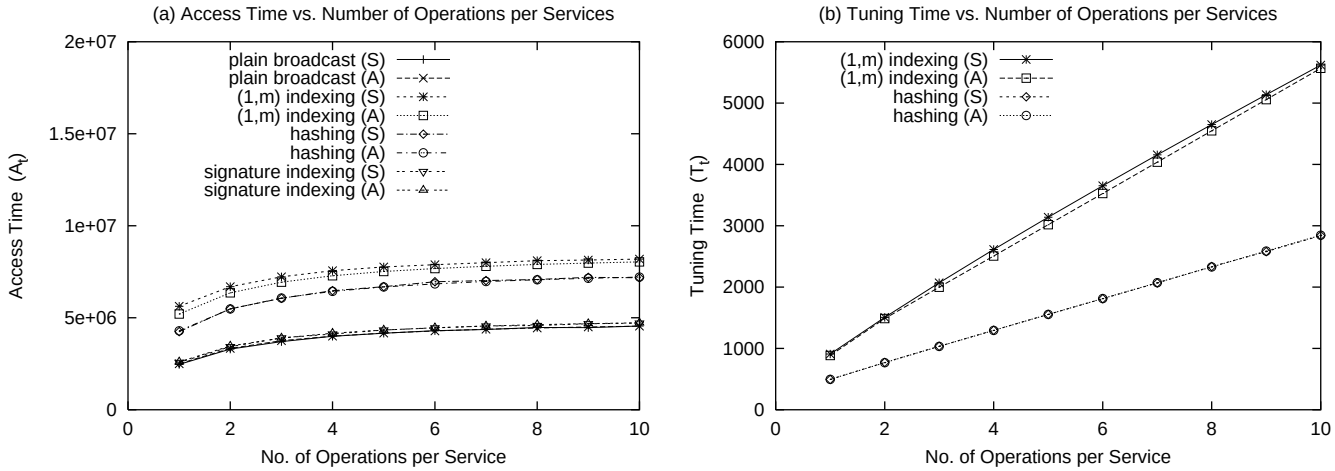

Fig. 10. Compare all access methods for data channel with varied number of operations

imizing power consumption and clients' waiting time. We presented a multi-channel model for m-services. Three types of information are accessible through those channels: advertised information about m-services (UDDI channel), m-services' descriptions and codes (m-services channel), and data needed by m-services during their execution (data channel). To enable efficient access to wireless channels, we extend three well-known indexing techniques to the context of m-services: B+ tree, signature indexing, and hashing.

To illustrate the viability of our approach, we conducted analytical and experimental analysis of the aforementioned techniques. From the experiments, we note the following observations: (1) Plain broadcast (no access method) has the best access time but the worst tuning time. Since the tuning time of plain broadcast is far larger than that of any access method, it is usually not a preferred method in power limited wireless environments; (2) Signature indexing is the most suitable access method for m-service channel since it exhibits better access time and tuning time performance than all other methods; (3) When broadcast information has very large size (e.g. m-services), using smaller broadcast buckets can increase the performance of hashing method; (4) Access methods can be applied to varied-size m-service channel without introducing considerable overhead. (5) In data channel, signature indexing achieves better access time than most of the other access methods schemes. However, the tuning time of signature indexing is pretty large. When energy is of little concern compared to waiting time, signature indexing is a preferred method. (6) Hashing achieves the best tuning time and good access time in data channel. For energy critical applications or at least when energy is not a negligible factor, hashing method will be the preferred method in data channel. The experiments showed that the best techniques for m-services and data channels are signature indexing and hashing methods respectively.

As future work, we propose to organize m-services and data into multiple channels. M-services would be grouped into different *communities* based on their category (e.g., airlines, hotels, finance). Each m-services channel would contain m-services (description and code) that belong to the same community. Similarly, each data channel would contain data accessible by m-services within a specific community. Combined with the proposed indexing techniques, such organization would accelerate access to m-services and data. Instead of looking into all m-services and data within the current geographic area, mobile users would focus on m-services that correspond to their category of interest.

Acknowledgment. The authors would like to thank he anonymous reviewers and Mourad Ouzzani for their valuable comments on earlier drafts of this paper.

REFERENCES

[1] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid, "Business-to-Business Interactions: Issues and Enabling Technologies," *The VLDB Journal*, vol. 12, no. 1, May 2003.

[2] R. Kalakota and A. B. Whinston, *Frontiers of Electronic Commerce*. Addison Wesley (ISBN: 0-201-84520-2), February 2000.

[3] Z. Maamar, B. Benatallah, and Q. Z. Sheng, "Towards a Composition Framework for E-/M-Services," in *First International Workshop on M-Services - Concepts, Approaches, and Tools*, Lyon, France, June 2002.

[4] J. A. Senn, "The Emergence of M-Commerce," *IEEE Computer*, vol. 33, no. 12, December 2000.

[5] U. Varshney and R. J. Vetter, Eds., *Special Issue on Mobile Commerce*, ser. MONET 7(3), June 2002.

[6] U. Varshney and R. J. Vetter, "Mobile Commerce: Framework, Applications and Networking Support," *MONET*, vol. 7, no. 3, June 2002.

[7] B. Medjahed, A. Rezgui, A. Bouguettaya, and M. Ouzzani, "Infrastructure for E-Government Web Services," *IEEE Internet Computing*, vol. 7, no. 1, January 2003.

[8] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 2, February 2002.

[9] B. Benatallah and F. Casati, Eds., *Special Issue on Web Services*, ser. *Distributed and Parallel Databases, an International Journal*, 12(2), November 2002.

[10] X. Yang and A. Bouguettaya, "Broadcast-based Data Access in Wireless Environments," in *8th International Conference on Extending Database Technology*, Prague, Czech Republic, March 2002.

[11] W3C, *Universal Description, Discovery, and Integration (UDDI)*, http://www.uddi.org.

[12] Z. Maamar, W. Mansoor, and H. Yahyaoui, "E-Commerce through Wireless Devices," in *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Cambridge, MA, USA, June 2001.

[13] W3C, *Simple Object Access Protocol (SOAP)*, http://www.w3.org/TR/soap.
[14] D. Dahlem, J. H. Jahnke, A. Onabajo, and O. Wang, "The Challenges of Implementing Web Services on Small Devices," in *OOPSLA Workshop on Pervasive computing: Going Beyond Internet for Small Screens*, Seattle, Washington, USA, November 2002.
[15] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy Efficient Indexing on Air," in *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, March 1994.
[16] ——, "Power Efficient Filtering of Data an Air," in *Proceedings of 4th International Conference on Extending Database Technology (EDBT'94)*, Cambridge, United Kingdom, March 1994.
[17] I. Akyildiz, J. McNair, J. Ho, H. Uzunalioglu, and W. Wang, "Mobility Management in Next-Generation Wireless Systems," in *Proceedings of the IEEE*, vol. 87, August 1999.
[18] W3C, *Web Services Description Language (WSDL)*, http://www.w3.org/TR/wsdl.
[19] G. F. Welch, "A Survey of Power Management Techniques in Mobile Computing Operating Systems," *Operating Systems Review*, vol. 29, no. 4, 1995.
[20] W.-C. Lee and D. Lee, "Using Signature Techniques for Information Filtering in Wireless and Mobile Environments," *Special Issue on Databases and Mobile Computing, Journal on Distributed and Parallel Databases*, vol. 4, no. 3, July 1996.
[21] WAP, *Wireless Access Protocol*, http://www.wapforum.org/what.
[22] Z. Maamar, W. Mansoor, and Q. Mahmoud, "Software Agents to Support Mobile Services," in *International Conference on Autonomous Agents and Multi Agents Systems*, Bologna, Italy, July 2002.
[23] U. Varshney, "Addressing Location Issues in Moble Commerce," in *IEEE Conference on Local Computer Networks*, Tampa, Florida, USA, November 2001.
[24] A. D. Malloy, U. Varshney, and A. P. Snow, "Supporting Mobile Commerce Applications Using Dependable Wireless Networks," *MONET*, vol. 7, no. 3, June 2002.
[25] L. D. Paulson, "Going Mobile with Web Services," *IEEE Computer*, vol. 36, no. 2, February 2003.

Xu Yang received B.E. degree from HuaZhong University of Science and Technology, China in 1996 and M.S. in Computer Science from Queensland University of Technology, Australia in 1998. He is a doctoral candidate in Computer Science at Virginia Tech. His research interests include mobile computing and Web Services, with particular emphasis on Wireless Access Methods, Wireless Indexing and Data Broadcast. He also works as a senior software engineer with Spirent Communications.

Athman Bouguettaya is Program Director of Computer Science at Virginia Tech. He is also Director of the E-Commerce and E-Government Research Lab at Virginia Tech. He is on the editorial boards of the Distributed and Parallel Databases Journal and the International Journal of Web Services Research. He guest co-edited a special issue of the IEEE Internet Computing on Database Technology on the Web. He served as the Program co-chair of the IEEE RIDE Workshop on Web Services for E-Commerce and E-Government (RIDE-WS-ECEG'04). He served on numerous conference program committees. He is the author of more than 60 publications. His latest research interests are in Web databases and Web services.

Brahim Medjahed is a PhD candidate in the Department of Computer Science at Virginia Tech. He received a BSc and MSc in computer science from USTHB University (Algeria) with First Honor. His research interests include Web services, workfbws, and Web databases. The focus of his PhD dissertation is on the automatic selection and composition of Web services on the Semantic Web. Brahim has published several papers on Web services in international journals and conferences including the VLDB Journal, IEEE Internet Computing, IEEE Computer, VLDB, SOC, WIDM, and TES. He was nominated as outstanding reviewer by IEEE Internet Computing. Brahim also serves in the program committees of workshops on ubiquitous computing and M-Services. He is student member of IEEE, IEEE Computer Society, and ACM.

Weiping He received the B.E. degree from Huangzhong University of Science and Technology (P. R. China) in 1996. M.S. in Computer Science from Virginia Tech in 2002. He is a doctoral candidate in Computer Science at Virginia Tech. His research interests include reliability in mobile computing and Web Services, with particular emphasis on failure recovery in M-Commerce. He is also the system administrator for the Computer Science department. He has considerable experience in system administration.

Hao Long received her Bachelor's degree in Aerospace Engineering from Beijing University of Aero. and Astro. (P. R. China) in 1996 and her Master's degree in Computer Science from Virginia Tech in 2002. After graduation, She had been working as a research assistant in E-Commerce and E-Government Lab in Virginia Tech for one year. Now she works for Georgetown University as programmer analyst. Hao has a special interest in leveraging the power of Web Services to provide cross-platform e-business solutions in both wired and wireless enviroment. She has extensive hands-on experience in Java, J2EE, XML, and Web Services (SOAP/WSDL/UDDI), to name but a few.