Air Indexing for On-Demand XML Data Broadcast

Weiwei SUN
Fudan University

Rongrui QIN
University of Adelaide

Jinjin WU
Fudan University

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Air Indexing for On-Demand XML Data Broadcast

Weiwei Sun, Yongrui Qin, Jingjing Wu, Baihua Zheng, Zhuoyao Zhang, Ping Yu, Peng Liu, Jian Zhang

Abstract—XML data broadcast is an efficient way to disseminate semi-structured information in wireless mobile environments. In this paper, we propose a novel two-tier index structure to facilitate the access of XML document in an on-demand broadcast system. It provides the clients with an overall image of all the XML documents available at the server side and hence enables the clients to locate complete result sets accordingly. A pruning strategy is developed to cut down the index size and a two-tier structure is proposed to further remove any redundant information. In addition, two index distribution strategies, namely naive distribution and partial distribution, have been designed to interleave the index information with the XML documents in the wireless channels. Theoretical analysis and simulation experiments are also put forward to show the benefits of our indexing methods.

Index Terms—XML data broadcast, air indexing, two-tier index, index distribution, on-demand broadcast.

1 Introduction

Wireless technology has become deeply embedded in everyday life. At the end of 2011, there were 6 billion mobile subscriptions, estimated by the International T-elecommunication Union (2011). That is equivalent to 87 percent of the world population, and is a huge increase from 5.4 billion in 2010 and 4.7 billion mobile subscriptions in 2009. According to ABI research, mobile application revenue hits 8.5 billion in 2011, and it will reach an estimated 46 billion in 2016. Obviously, the market for mobile applications will become "as big as the internet" in the near future.

Fundamentally, mobile information access via various wireless technologies (e.g., Bluetooth, WiFi, Satellite) can be classified into two basic methods: *point-to-point access* and *broadcast*. Point-to-point access employs a pull-based approach where a mobile client initiates a data access request to the server which then processes the requirement and returns the requested data to the client over an exclusive point-to-point wireless channel. It is suitable for lightly loaded systems in which wireless channels and server processing capacity are not severely contended. However, as the number of requirements increases, the system performance is deteriorated as the server has to create a point-to-point channel for each requirement.

Different from point-to-point communication, wireless broadcast requires the server to push data to the clients over a dedicated broadcast channel. It allows an arbitrary number of clients to access data simultaneously and thus is particularly suitable for heavily loaded systems or the systems in which the users share similar data access patterns. For example, during London 2012 Olympics, a wireless broadcast system can update the mobile users with live results, calendar schedule, and medal tables. Another example is that a museum/gallary may broadcast its hours, current attractions, and profiles of artworks to its visitors, whereas point-to-point access is used only for fulfilling occasional, ad hoc requests.

Wireless data broadcast services have been available as commercial products for many years, e.g. StarBand and Hughes Network. Recently, there has been a push for such systems from the industry and various standard bodies. For example, born out of the ITU "IMT-2000" initiative, the Third Generation Partnership Project 2 is developing Broadcast and Multicast Service in CDMA2000 Wireless IP network. Systems for Digital Audio Broadcast (DAB) and Digital Video Broadcast (DVB) are capable of delivering wireless data services. Recent news also reported that XM Satellite Radio (www.xmradio.com) and Raytheon have jointly built a communication system, known as the Mobile Enhanced Situational Awareness (MESA) Network, that would use XMs satellites to relay information to soldiers and emergency responders during a homeland security crisis.

Depending on whether or not the client has the ability to submit requests, broadcast methods can be divided into two categories, *push-based* broadcast and *on-demand* broadcast. For push-based broadcast, the client does not submit any request to the server but just listens to the broadcast channels and it is the client's responsibility to find the answers to its requests. For on-demand broadcast, clients can submit their requests to the server like in the ordinary on-demand access mode, while the server transfers information to clients on public broadcasting channels.

On the other hand, more information has been available as semi-structured or unstructured data over the

- *Weiwei Sun, Jingjing Wu, Zhuoyao Zhang, Ping Yu, Peng Liu, and Jian Zhang are with School of Computer Science, Fudan University. E-mail: {wwsun, wjj, zhangzhuoyao, pingyu, liupeng, zhang_jian}@fudan.edu.cn*
- *Baihua Zheng is with School of Information Systems, Singapore Management University. Email: bhzheng@smu.edu.sg*
- *Yongrui Qin is with School of Computer Science, The University of Adelaide, Adelaide, SA 5005, Australia. Email: yongrui.qin@adelaide.edu.au*

past few years. XML holds the potential to become the de facto standard for data integration. The fact is that most Internet browsers provide support for XML in their later versions and nearly all the major IT companies (e.g., Microsoft, Oracle, and IBM) have integrated XML into the software products which further demonstrates the popularity of XML. In the near future, XML documents, like HTML Web pages, might become a part of our daily life. Consequently, XML has attracted attentions from database community recently and there has been a large body of research work focusing on XML, such as XML filtering, querying and indexing [1], [2], [3].

We focus on the access of XML documents in wireless on-demand broadcast systems in this paper. We propose a novel two-tier indexing approach and a partial index distribution to facilitate the dissemination of XML documents in on-demand wireless broadcast environments. Our target is a general and scalable on-demand broadcast system that can support requests on a large XML documents collection from a huge client population.

In our preliminary study [13], we proposed a novel two-tier index structure to improve the index size as well as tuning time performance. In this paper, we extend our previous work in the following aspects:

- Two index distribution strategies, namely naive distribution, and partial distribution, have been proposed to interleave the index information with the XML document over the wireless channels.
- A cost model has been proposed to achieve a better trade off between access time and tuning time in order to satisfy different system requirement.
- A more comprehensive experimental study has been performed. First, we use larger datasets. Compared with the dataset used in our preliminary study that contains only 1000 XML documents, the datasets used in our new study contain a larger number of XML documents, ranging from 100 to 100,000. Second, new sets of experiments have been conducted to compare the performance of proposed two-tier structure against existing XML indexes, and to report the space requirement and construction cost of two-tier index structure.

The rest of the paper is organized as follows. Section 2 presents the preliminaries of the problem. Section 3 shows the overview of our work. Section 4 describes the two-tier indexing method, including the pruning technology and the two-tier structure. Section 5 explains index distribution strategies, together with theoretical analysis. Section 6 explains the simulation model and reports the performance evaluation results. Finally, Section 7 concludes our work.

2 Preliminaries

In this section, we first describe the on-demand data broadcast model and list all the assumptions. Thereafter, we briefly review related air indexing techniques.

2.1 On-demand data broadcast

In this paper, we adopt a general system model, as depicted in Fig. 1. It consists of three parts: 1) the communication mechanism; 2) the broadcast server; and 3) the mobile clients. Our target is to deliver XML documents according to user requests efficiently. We assume that users submit their requests in the form of XPath queries to the server over the uplink channel, and then wait for the result documents via listening to the broadcast channel. The server, after receiving user queries, identifies the result documents and calls the scheduler to arrange the broadcast program based on different scheduling algorithms. It is noticed that scheduling of XML documents is not the focus of our work and we assume the server invokes some existing algorithms (e.g. [4], [5]) to generate the broadcast program.

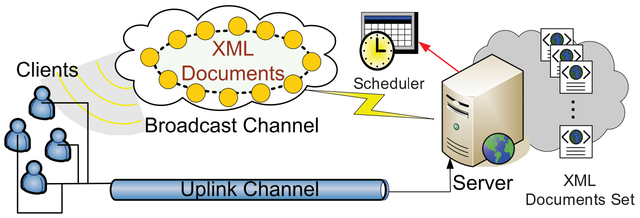


Fig. 1. A general system model

We adopt on-demand broadcast in this paper to improve the usability of the wireless channel. The reasons are two-fold. First, on-demand mode can adapt to the change of client's access patterns, since the organization of each broadcast cycle is dynamically determined by the server based on the current pending queries. Second, on-demand mode can utilize the scarce bandwidth more efficiently because the users are usually only interested in a small portion (e.g., 1%) of the entire document collection, and documents not requested by users will not be broadcast in this mode.

We assume the server accumulates the user requests (i.e., XPath queries) in a local queue and answers part of the pending queries in each broadcast cycle based on the scheduling strategy. Without loss of generality, we assume that a result set of an XPath query contains one or multiple XML documents and an XML document might satisfy multiple XPath queries. In addition, the result set for each request is not empty and the user aims at retrieving all the result XML documents.

As mobile clients are typically powered by batteries with limited capacity, the main concerns of a wireless broadcast system are i) how fast a request could be satisfied; and ii) how battery energy of mobile clients could be conserved. Accordingly, two performance metrics, namely *Access Time (AT)* and *Tuning Time (TT)*, are commonly used to measure access efficiency and energy consumption for mobile clients in a wireless data broadcast system [6], respectively. The former is the time elapsed from the moment a query is issued to the moment it is answered and the latter is the time a mobile client stays in active mode to receive the requested XML documents and index information.

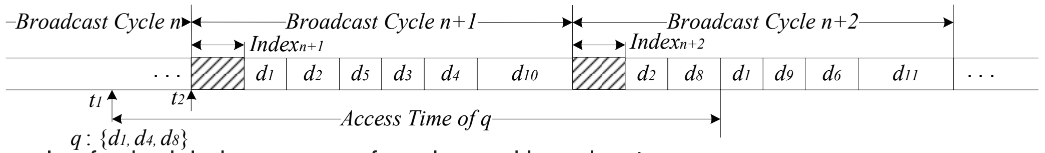


Fig. 2. An example of scheduled sequence of on-demand broadcast

2.2 Air Indexing

Air indexing techniques are often used for conserving the energy of mobile clients [7]. The basic idea is that the broadcast server pre-computes indexing information (including searchable attribute and delivery time of data objects) and interleaves it with data objects (e.g., XML documents) in the broadcast channel. By examining the index, mobile clients are aware of the arrival time of desired data objects and know when to switch to active mode for data retrieval. Similarly, appending to each data object, the delivery time of the next index helps the clients to schedule the sleep time for sub-sequential data access. Some well-known air indexes are reviewed in Appendix-A.

Fig. 2 shows an example of a broadcast program. Within each broadcast cycle, the server broadcasts the index first and then the XML documents. With the help of index, the client can access the XML documents in the following steps. First, in *Initial Probe* step the client sends a query to the server, and then tunes into the channel to find out the arrival time of the next index. Second, in *Index Search* step the client tunes into the channel to retrieve the index and find out the broadcast time of the result documents. Then, in *Document Retrieval* step the client downloads the result documents when they are broadcast. The client repeats the index search step and document retrieval step until all the result documents are retrieved. Appendix-B illustrates how a query is processed at client side to facilitate the understanding of XML document retrieval.

After explaining the client side process, we are ready to discuss server side process. In order to support on-demand XML data broadcast, the server needs to schedule the broadcast program and decide the content of the index. As we focus on air indexing in this paper, we assume the server treats each XML document as a data item and adopts some existing scheduling algorithms for data broadcast. As for the index structure, a simple approach is to adjust existing XML index schemes to the broadcasting environments. However, those index schemes are designed for normal storage media (e.g., hard disks) and are usually big in size [8], [9], [10], [11], [12]. Given the fact that the bandwidth of a wireless channel is limited and the storage capacity of mobile users is very low, those index schemes are not suitable for XML data broadcast. On the other hand, there are some works on wireless XML streaming [14], [15], [16], [17]. They assume the mobile clients keep listening to the wireless channel and processing the XML data continuously. Those schemes enable the mobile clients to receive the data without sending queries to the server. In other words, those index schemes do not provide an overall picture of the XML data and hence the clients do not know whether all the matched XML data have been retrieved. Consequently, they are not suitable for on-demand XML broadcast. We will demonstrate the inefficiency of those works in Section 6.

3 OVERVIEW OF OUR WORK

In this section, we sketch the main idea of our approach, with the architecture of our air indexing method depicted in Fig. 3. There are two main components included, namely *index constructor* and *index distributor*, whose functionalities will be detailed later.

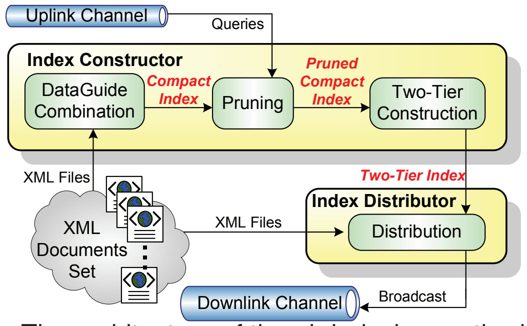


Fig. 3. The architecture of the air indexing method

The index constructor is responsible for building the index structure to support on-demand XML data broadcast, and it involves three sub-tasks as shown in Fig. 3. First, we adopt an existing XML index scheme (e.g., DataGuides [3] in our implementation) to index the structural information of each XML document in the collection. The indexes corresponding to individual XML documents are combined to form *compact index*. Second, pruning techniques are applied to remove unnecessary nodes from the index according to the current query set pending at server side, and the resulting index is named as *pruned compact index*. Third, a two-tier index is built to further improve the system performance via eliminating the common pointers shared by multiple requests, and we will detail the index structure in Section 4.

Once the index is constructed, the index distributor starts the distribution step to interleave the index information and XML documents and form the broadcast program. A simple approach is to broadcast the index in the beginning of each broadcast cycle so that the clients can locate all the qualified documents (i.e., IDs and arrival time of qualified documents) by listening to one index. However, as the index is constructed based on all the pending requests whose number might be large, its size could be very big which will extend the broadcast cycle and hence prolong the access latency. Alternatively, we propose an efficient index distribution strategy, namely *partial distribution*, that strikes a balance between access time and tuning time performance. Section 5 will present the index distribution strategies in detail.

4 THE TWO-TIER INDEXING METHOD

In this section, we introduce a novel two-tier index structure that is based on the combined DataGuides [3]. We first introduce the concept of combined DataGuides which captures the path information of all the documents. Then, we present the pruning techniques to further reduce the index size in an on-demand environment. Finally, we introduce the two-tier structure. To facilitate our discussion, a sample XML document collection is depicted in Fig. 4, and a sample query set is listed in Table 1. Although the sample XDL document collection contains only five XML documents, we want to highlight that our system has excellent scalability and it can support various XML document collections containing a large number of documents (e.g., $100,000$).

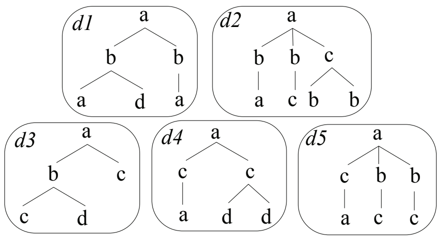


Fig. 4. Sample XML document set

TABLE 1
Sample XPath query set

Q	ID list of matched documents
q_1:/a/b/a	d_1, d_2
q_2:/a/c/a	d_4, d_5
q_3:/a//c	d_2, d_3, d_4, d_5
q_4:/a/b	d_1, d_2, d_3, d_5
q_5:/a/c/*	d_2, d_4, d_5
q_6:/a/c/a	d_4, d_5

4.1 Combined DataGuides

As each DataGuide (refer to Appendix-C for a brief introduction of DataGuide) is built for one single XML document, we adopt RoxSum [18] to integrate DataGuides of all the XML documents in collection into a *Compact Index*. Fig. 5 shows the integrated version of all the DataGuides related to the sample XML documents. The compact index enables path sharing among a set of XML documents and hence significantly reduces the index size. For our sample XML documents, the compact index has only 9 nodes while the original DataGuides contain in total 24 nodes.

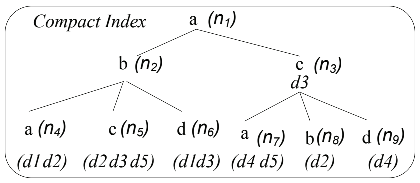


Fig. 5. The Compact index

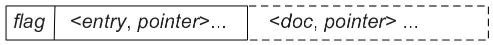


Fig. 6. The structure of an index node

Given the compact index, we propose a three-block index structure, as depicted in Fig. 6, to carry the information. The first block contains a flag to indicate the type of the corresponding index node. There are three kinds of nodes in the compact index, i.e., *internal nodes* with flag = 0, *leaf nodes* with flag = 1, and *root nodes* with flag set to the real index values. The second block contains $\langle entry, pointer\rangle$ tuples, carrying the child nodes' information. The last block is made up of $\langle doc, pointer\rangle$ tuples pointing to real XML documents which are going to be broadcasted. A node might not necessarily have all three blocks. For example, a leaf node only has flag and $\langle doc, pointer\rangle$ tuples, while an internal node usually only has flag and $\langle entry, pointer\rangle$ tuples. Note that, an internal node of the compact index may also have $\langle doc, pointer\rangle$ tuples in addition to $\langle entry, pointer\rangle$ tuples, e.g., n_3 in Fig. 5. The detailed contents as well as the broadcast order of all the nodes are depicted in Fig. 7.

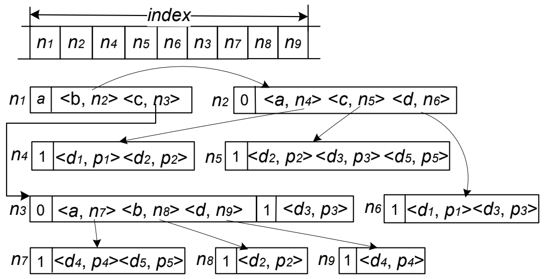

index
n1 n2 n4 n5 n6 n3 n7 n8 n9
n1 a <b, n2> <c, n3>
n2 0 <a, n4> <c, n5> <d, n6>
n4 1 <d1, p1> <d2, p2>
n5 1 <d2, p2> <d3, p3> <d5, p5>
n3 0 <a, n7> <b, n8> <d, n9> 1 <d3, p3>
n6 1 <d1, p1> <d3, p3>
n7 1 <d4, p4> <d5, p5>
n8 1 <d2, p2>
n9 1 <d4, p4>

Fig. 7. The index of sample XML document

With the help of the compact index, query processing is straightforward. Take query q_1, that asks for documents satisfying the XPath query /a/b/a, as an example. Suppose that the client knows the start of the index and tunes into the channel when n_1 is to be broadcast. It first retrieves n_1. Since the value of n_1 is 'a' which matches the first part of q_1, it exams $\langle entry, pointer\rangle$ to look for the entry with value 'b' (i.e., n_2). Then, it follows the pointer to access n_2 and similarly later retrieves n_4. As node n_4 is a leaf node, the result documents are located, i.e., d_1 and d_2. The client can then tune into doze mode for energy saving and tune into the channel only when d_1 or d_2 is to be broadcast. The query can be satisfied when all the result documents are downloaded.

4.2 Index pruning

It is noticed that the compact index is built based on the entire document collection. However, in on-demand broadcast mode, the requested document set could be much smaller than the original collection. According to the Pareto principle, roughly 80% of the users only ask for 20% of the documents. Consequently, many index nodes actually point to the documents not requested by any user. Motivated by this observation, we propose a pruning strategy to eliminate all the dead nodes that will not be accessed by any pending query.

The pruning process works as follows. Given the compact index, all the paths that match at least one query pending at the server side are marked. Thereafter, a *Pruned Compact Index* containing only the marked nodes is built. It is noticed that when a node n has some of its child nodes pruned, and meanwhile the path from the root node to node n actually satisfies some queries, node n has to check the documents associated with the pruned child nodes as well to avoid the loss of qualified document information caused by node pruning. It has to re-associate with itself all the documents that are *exclusively* associated with any of its pruned child nodes. In case the pruned child node is not in the leaf level, the document re-association has to be propagated from the leaf level to its parent node and finally reaches node n. The pruning algorithm is described in Algorithm 1 of Appendix-D, together with an explanation of its implementation issues.

We use an example to explain the pruning process. We assume the query set Q pending at the server side is $\{$/a/b, /a/b/c, /a//c$\}$. Fig. 8 is the pruning result of the original compact index (shown in Fig. 5). It is noticed that only nodes n_1, n_2, n_3 and n_5 match the queries, and the rest nodes are discarded.

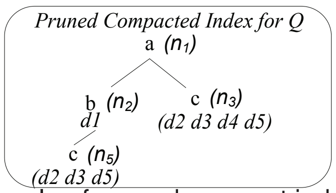


Fig. 8. An example of pruned compact index

4.3 The two-tier index structure

As we mentioned in Section 2, an XML document might satisfy multiple queries and hence it is included in multiple result document sets. Take document d_2 as an example. It matches the XPath queries a/b/a, a/b/c and a/c/b. As a result, the tuple $\langle d_2, p_2\rangle$ appears three times in the index (see Fig. 7). In order to reduce the duplication, we propose a two-tier index structure which removes the pointers to the result documents from the current index structure and packets them into a second-tier document pointer list. Specifically, instead of maintaining $\langle doc, pointer\rangle$ tuples in the compact index, the two-tier index only stores the IDs of all the result documents in the format of $\langle d_i, d_j, \cdots, d_k\rangle$ tuple in the first tier. The pointer information of documents is moved to the second-tier in the format of $\langle doc, pointer\rangle$ tuple. Fig. 9 depicts the new placement of the pruned compact index of the running example, with its one-tier structure shown in Fig. 7. We explain the theory behind the design of the two-tier structure in Appendix-E.

With the two-tier structure index, the query processing involves two phases, i.e., finding the result documents and retrieving the result documents. Back to $q_1 = /a/b/a$ as an example. First, the client retrieves the first-tier index to find the document IDs of all the result documents, i.e., d_1 and d_2. Then, it listens to the second-tier

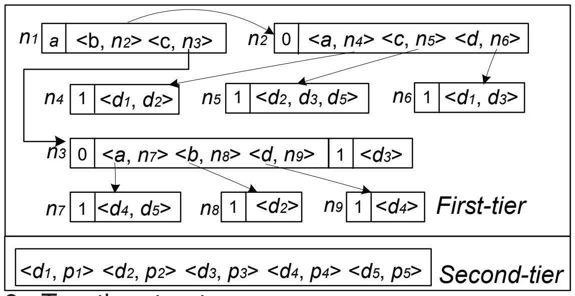


Fig. 9. Two-tier structure

to find out when d_1 and d_2 will be broadcast, and follows the pointers to download the documents.

Although the major motivation of the two-tier structure index is to eliminate the redundant pointer information in the index, the separation of the identification of result documents and the pointer information can bring other advantages for the XML broadcast system. For example, with the two-tier index, the client needs to access the first-tier index only once to find the IDs of the matched documents, no matter when the result documents are scheduled to broadcast. Consequently, the client only needs to listen to the second tier index in the following broadcast cycles until all the result documents are retrieved, which significantly improves the tuning time performance. In addition, the two-tier structure offers great flexility for the distribution of the index, and its benefit will be further demonstrated in Section 5 where we introduce the index distribution strategies.

5 INDEX DISTRIBUTION

Once the index is constructed, the server needs to disseminate it in the wireless channel. We propose two s-trategies, namely *naive distribution* and *partial distribution*, to interleave the index information with XML documents in wireless channels. The former simply distributes the index once in the beginning of each broadcast cycle. This approach is straightforward and it guarantees that the client can find out the complete document set in terms of IDs by listening to the index once. However, it has to broadcast the entire index in every broadcast cycle even most of the documents indexed might not be available during the current cycle. Consequently, this approach suffers from a prolonged broadcast cycle which actually extends clients' access time. In order to address this issue, a partial distribution strategy is proposed which tries to strike a balance between tuning time and access time. Table 2 lists notations used in the following description. Note that the formal definitions of complete index, partial index, and broadcast round will be presented in Section 5.2.

5.1 Naive distribution

We assume the naive index is constructed based on the *entire* XML document collection and pruned by the queries pending at the server side right before the broadcast of the current cycle. As the index enables the clients who submit the queries before the beginning of the current broadcast cycle to locate the IDs of all the result documents, it is named as *complete index* to be distinguished from the index used by the partial distribution to be introduced in Section 5.2. Concluded by Appendix-F in which query processing of naive distribution is discussed as well as the pseudo-code, the average tuning time, denoted as TT_n, can be approximated in Equation (1).

$$TT_n = L_I + c_n \times L_{II} + d_{xml} \times r_n \tag{1}$$

Note that we use the size of the information needed to be accessed to represent the tuning time. We ignore the analysis of the average access time AT_n as it is highly dependent on the scheduling results.

5.2 Partial Distribution

Given the fact that only a small collection of documents indexed by the complete index are broadcast within each cycle, broadcasting the complete index of those unavailable documents in the current cycle actually wastes bandwidth. Consequently, a new index distribution strategy, namely *partial distribution*, is proposed. It builds a *partial index* based on the documents that will be distributed in the current cycle to reduce the index size and hence the access time. Partial index is usually much smaller than complete index, which means more XML data could be broadcast within a cycle given that the cycle size is fixed. However, the partial index only gives a local view of the documents broadcast within each broadcast cycle, and it fails to provide the clients with a global view of the entire document collection. By only listening to the partial indexes, the client cannot assert whether all the required documents matching their queries have been identified. They may have to continuously listen to the wireless broadcast channel which definitely extends the access time.

The solution we propose in this paper is to introduce a new concept, namely *broadcast round* that is a collection of m adjacent broadcast cycles. Within each broadcast round, the complete index built based on the entire XML document collection is broadcast once, in the beginning of the first broadcast cycle within the broadcast round. Thereafter, in the beginning of the rest $(m-1)$ broadcast cycles within the broadcast round, a partial index is broadcast. Notice that a partial index corresponding to a broadcast cycle is built based on the documents broadcast within that broadcast cycle only.

TABLE 2
Notations for two-tier index

Notation	Description
L_I	the size of the first-tier of a complete index
L_I^p	the size of the first-tier of a partial index
L_{II}	the size of the second-tier index
L_D	the size of the documents broadcast in the first cycle in a broadcast round
L_D^p	the size of the documents broadcast in non-first cycle in a broadcast round
L_C	the size of a broadcast cycle
d_{xml}	the average size of an XML document
r_n	the average cardinality of a result set
c_n	the average number of cycles broadcast between the clients' starting retrieving the first index and the time when all the result documents are retrieved
m	the number of broadcast cycles within one broadcast round

We present the detailed client side query processing under partial distribution in Appendix-G.

Since we broadcast the complete index once every m broadcast cycles, the value of m has a direct impact on the system performance. In order to quantify m's impact, we conduct a theoretical study in the following. We would like to highlight that we assume the broadcast cycle generated by the server is only dependent on the scheduling algorithm and the pending queries, but not the index distribution strategy. In other words, the broadcasting order of XML documents is fixed.

First, we analyze the average tuning time under the partial index distribution, denoted as TT_p. As shown in Equation (2), it involves four components, which are i) The retrieval time of the first-tier of the partial index; ii) The retrieval time of the first-tier of the complete index: the processing of the query cannot be completed until the first-tier of a complete index is retrieved; iii) The retrieval time of the second-tier index: the client needs to access on average c_n second-tier index; and iv) The retrieval of the result XML documents.

$$TT_p = \frac{m-1}{2} \times L_I^p + L_I + c_n \times L_{II} + d_{xml} \times r_n \tag{2}$$

Under our assumption that the broadcast order of XML documents is fixed, the access time is determined by the bandwidth occupied by the index within each cycle. The larger the index is, the more the bandwidth it occupies. Consequently, less bandwidth is available for dissemination of XML documents and longer access time is expected. Based on this observation, we can conclude that average access time is proportional to the bandwidth allocated to data and it can be estimated based on Equation (3). Here, $AvgB_p/AvgB_n$ stands for the average bandwidth allocated to broadcast data under partial/naive index distribution. To standardize the comparison, we use the amount of data broadcast within one broadcast round.

$$AT_p/AT_n \approx \frac{AvgB_p}{AvgB_n} = \frac{m \times L_D}{L_D+(m-1)L_D^p} \tag{3}$$

$$AT_p \approx \frac{m \times L_D}{L_D+(m-1)L_D^p} \times AT_n \tag{4}$$

Based on Equation (2) and the deduction presented in Appendix-H, we understand that a large m favors access time but results in long tuning time. On the contrary, a small m helps to improve the tuning time but extends the access time. In order to consider both the tuning time and the access time, Equation(5) is adopted as a cost function to evaluate the impact of m on the system performance, with w indicating the importance of access time with regard to tuning time. For example, $w = 1$ indicates that access time and tuning time are equally important, and $w = 0$ means only access time is counted.

$$Cost = w \cdot TT_p + AT_p \tag{5}$$

Given a constant w, the optimal setting of m is approximated in Equation (6). Please refer to Appendix-I for the detailed derivation, and we will verify its correctness in our simulation.

$$opt\_m \approx \sqrt{\frac{2 \cdot AT_n}{w \cdot L_C}(\frac{L_I - L_I^p}{L_I^p})} \tag{6}$$

6 Experiments and Evaluation

In this section, we first evaluate the effectiveness of our indexing method, including the pruning techniques and the two-tier structure, and then compare the two-tier index scheme with the existing works. Finally, we evaluate the performance of the index distribution strategies.

Similar to existing works [1], [2], we use simple XPath queries in our experiments[1]. A simple query can be expressed as a path expression of format $Q = /N|//N|QQ$ with $N = E|*$. Here, E is the element label, / denotes the child axis, // denotes the descendant axis, and * is the wildcard. We use News Industry Text Format (NITF) DTD and XML Documents generated using the IBM's Generator tool [19]. We implement the modified version of the generator [2] to generate synthetic XPath queries without predicates, and use YFilter to filter the documents and generate the ID list of matched documents for each query. In our study, we also evaluate the performance of our approaches based on NASA document set. As the findings are pretty much the same, we only report the results of NITF.

We treat XML documents as data items and adopt the algorithm proposed in [20] as the underlying scheduling algorithm to generate broadcast programs. The average length of a broadcast cycle is 5% of the size of the document set. In other words, the average number of the documents broadcast within a cycle is about 5% of the number of the total documents. The average size of an XML document is 12.8KB. We allocate two bytes to represent an ID of an XML document and four bytes to represent a pointer. Table 3 summarizes the system parameters. Please refer to Appendix-J for a detailed explanation of the simulation settings.

It is noted that for a given scheduling algorithm, the broadcast of XML document is independent on the index structure. In other words, no matter which kind of index is adopted, the time used to retrieve the documents is constant. Thus, in the following simulations, we only present the tuning time incurred during index search without considering the document retrieval. In addition, we use the number of bytes retrieved to represent the tuning time.

6.1 Evaluation of indexing design

We have proposed two techniques to improve the proposed index structure, i.e., index pruning that prunes away index nodes corresponding to non-requested documents and a two-tier structure that removes the duplicated pointers to the XML documents. Their effectiveness is evaluated as follows.

1. All these experiments are simulated in Windows 7 of Pentium Dual-Core CPU E5400 4G RAM.

TABLE 3
Experiment setup

Variable	Description	Default value
N_D	The number of XML documents	10000
N_Q	The number of queries submitted to the server during the broadcasting period of each cycle	10000
P	The probability of wildcard * and double slash // in queries	0.1
D_Q	Maximum depth of queries	10

6.1.1 *Effectiveness of index pruning*

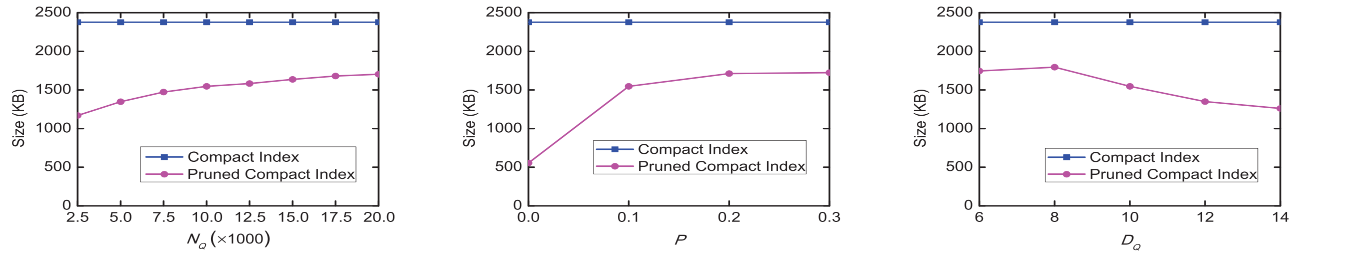
The pruning technology can reduce the index size. First, we compare the index size under different query set size N_Q settings, as reported in Fig. 10(a). We observe that even the compact index before pruning is small (around 2.3MB), only around 1.8% of the size of the complete XML document set. The pruning technology can further reduce the index size, with the size of pruned compact index on average about 56.5% of that of compact index. As N_Q increases, more XML documents are requested and hence the index becomes larger, which is consistent with our expectation. However, we can observe that the index size increases at a very slow pace. For example, when N_Q increases from 2.5K to 20K, the increase of the index size is less than 50%. This is because the impact of N_Q is considerably reduced by index combination and index pruning which in turn proves the effectiveness of our approach.

Second, we evaluate the index size under different wildcard and double slash probability (P) settings, as depicted in Fig. 10(b). We find that the size of compact index remains unchanged with the increase of P because it is mainly determined by the document set and is independent of the queries. On the other hand, the size of pruned compact index is proportional to P because a higher possibility of * and // in the query set implies that more documents will be included in the result set which deteriorates the efficiency of the index pruning.

Finally, we study the index size with different query depth D_Q values, and the result is shown in Fig. 10(c). When D_Q increases, the index sizes of pruned compact index get shrunk. The reason is that, a larger D_Q implies a smaller query selectivity which in turns results in a smaller result set for each query. On average, the pruned compact index can reduce at least 25% index size in most, if not all, the cases, compared with the compact index.

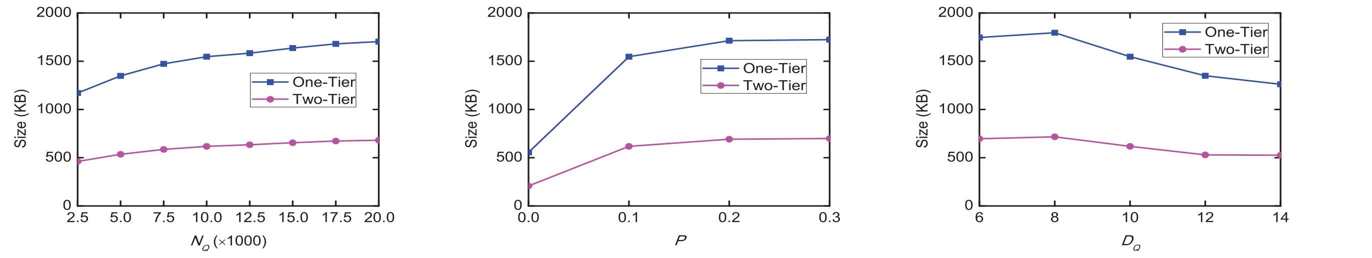
6.1.2 *Effectiveness of two-tier structure on index size*

Our second set of experiments is to evaluate the savings in terms of index size brought by the two-tier structure, with the result reported in Fig. 11. It is observed that two-tier structure significantly reduces the index size under different N_Q, P, D_Q. It implies a better access time and tuning time. From the above experimental results, we can prove the effectiveness of both the pruning technique and the two-tier structure. To be more accurate, the final index size can be reduced to 0.3%-0.6% compared with the data size.



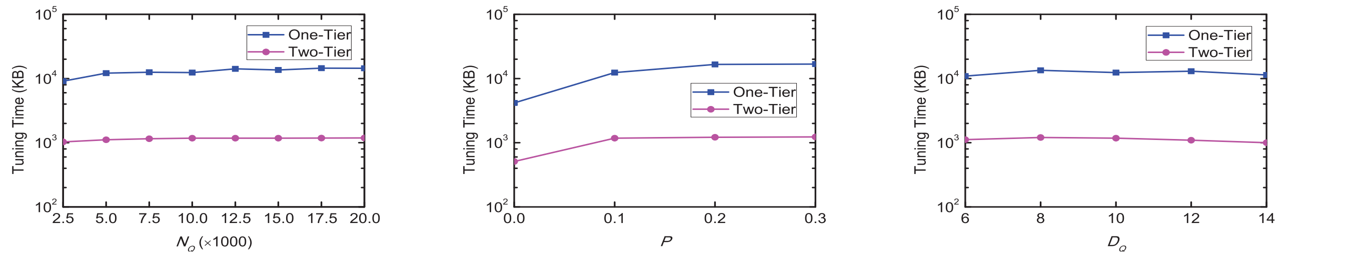
(a) Number of queries (b) Probability of * and // (c) Maximum depth of queries

Fig. 10. Effect of index pruning



(a) Number of queries (b) Probability of * and // (c) Maximum depth of queries

Fig. 11. Comparison of index size



(a) Number of queries (b) Probability of * and // (c) Maximum depth of queries

Fig. 12. Tuning time under one-tier vs. two-tier structures

6.1.3 Effectiveness of two-tier structure on tuning time

Our third set of experiments is to evaluate the savings in terms of the tuning time brought by the two-tier structure. When the index is broadcast as a one-tier structure, the client has to retrieve multiple index until all the result documents are retrieved. In our settings, each client has to listen to 33 broadcast cycles on average to complete one query. Consequently, if the index size is not small, this approach forces the clients to waste lots of energy accessing the index information repeatedly. Alternatively, the client only needs to access the second-tier index to retrieve the result documents if the document list is available in the second-tier.

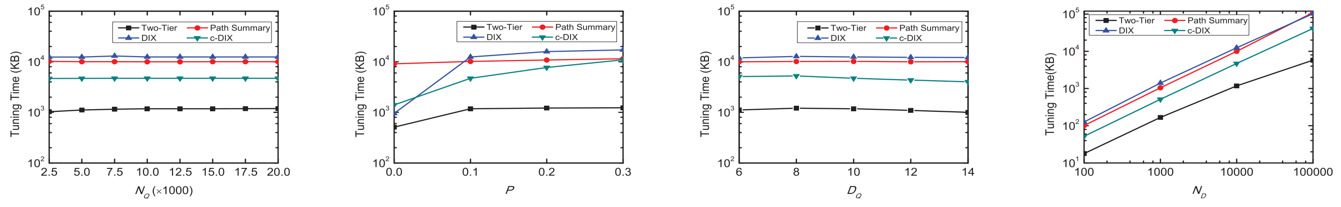
The simulation result is depicted in Fig. 12, with different N_Q, P, and D_Q settings. As expected, two-tier structure outperforms one-tier structure significantly. We also observe that parameters N_Q, P, and D_Q have a less significant impact on two-tier structure, compared with one-tier structure. The reason behind is that as N_Q, or P, or D_Q increases, more result documents are included into the answer set, and hence more cycles are expected to be accessed in order to finish one request. Clients need to download the first-tier index once under the two-tier structure and then listen to the second-tier index. Because the second-tier index is much smaller than the first-tier index, the access of the first-tier index dominates the tuning time and hence retrieving several more second-tier index does not change the tuning time a lot. On the other hand, access of first-tier index is time-consuming which explains the significant increase of the tuning time for the one-tier structure with the growing of N_Q, P, or D_Q.

6.1.4 Comparison of two-tier structure and existing XML indexes

We compare the performance of the two-tier structure against existing XML indexes of wireless XML streaming (i.e., Path Summary [14], DIX and clustered-DIX [15]). Please refer to Appendix-K for a brief introduction of Path Summary, DIX, and clustered-DIX.

First, we compare the index size. Since Path Summary, DIX and clustered-DIX are independent of queries, the result of index size under different variables (i.e. varying N_Q, P and D_Q) is the same. Consequently, we only compare their sizes against that of our two-tier index under the default settings as listed in Table 3. The index sizes of Path Summary, DIX and clustered-DIX under the default settings are 13.09MB, 18.49MB, and 23.71MB respectively, while that of our two-tier index is only about 0.62MB. This verifies our statement made in Section 1 that traditional XML indexes designed for normal storage media are big in size.

Next, we evaluate their tuning time performance in-



(a) Number of queries (b) Probability of * and // (c) Maximum depth of queries (d) Number of documents

Fig. 13. Comparison of the index tuning time of different XML index schemes

curred by the index search, as reported in Fig. 13. It is observed that our two-tier index performs consistently much better than the existing XML index schemes. The small index size contributes to the short tuning time performance. Besides, our two-tier index also has a good scalability in varying number of documents from 100 to 100,000 in Fig. 13(d). With the increasing of documents collection size, both the number of matched documents per query and the index size grow. Consequently, each client has to download a bigger index that results in a longer tuning time. However, we notice that compared with other index structures, the size of documents collections has a less significant impact on our two-tier structure. This is because a bigger documents collection actually leads to a more compact index that explains why two-tier structure increases the tuning time performance at a slower pace, compared with others.

6.2 Evaluation of index distribution strategies

In this set of experiments, we evaluate the performance of naive distribution and partial distribution strategies. We also verify our theoretical model developed in Section 5 through the experiments. It is noticed that naive distribution is a special instance of partial distribution with $m = 1$.

First, we compare the optimal m values derived based on theoretical analysis (in Equation (6)) with that obtained based on real experiments with $w = 5$, under different N_D settings. The result is shown in Table 4 with N_Q equals to N_D in each experiment. Though the theoretical optimal m is not exactly the same as the real one, the system performance (i.e., the $Cost$ function defined in Equation 5) under the theoretical optimal m is actually approaching the real optimal system performance. The system performance under theoretical optimal m is at most 0.3% larger than the real optimal performance.

TABLE 4
Optimal m values

N_D	Analysis Results		Simulation Results	
	opt_m	*cost* ($\times 10^6$)	*opt_m*	*cost* ($\times 10^6$)
2000	10.1	47.722	9	47.613
4000	10.1	94.144	9	94.106
6000	10.9	142.729	8	142.337
8000	10.0	184.611	8	184.432
10000	10.6	229.517	8	229.260

Second, we present the cost under partial distribution (w = 5) with different m values, and report the result in Fig. 14. The result of naive distribution is the same as that under partial distribution with $m = 1$. We can observe that the cost under partial distribution is improved first as m grows, and then gets increased. This is because initially as m increases, the client benefits more from the reduced index size. However, as m becomes too large, the benefit received from the reduced index size does not pay off the overhead of the extended tuning time performance. Here, the overhead includes longer access time and longer tuning time. The former is caused by the less frequent broadcast of the complete index, and the latter is caused by more index retrieval.

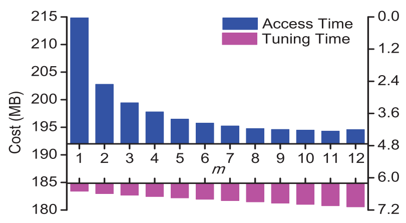


Fig. 14. Cost of *partial-index* under different m values (w=5, $N_D = 10000$)

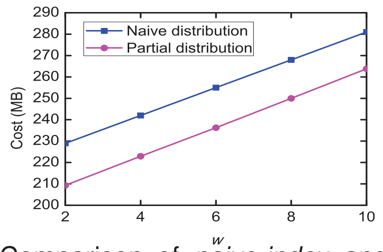


Fig. 15. Comparison of *naive-index* and *partial-index* under different w values

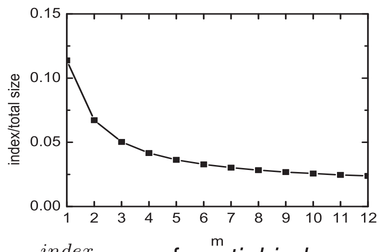


Fig. 16. $\frac{index}{broadcast\ round}$ of *partial-index* under different m values (w=5)

Third, we evaluate the performance of different index distribution under various w value, with results reported in Fig. 15. Note that we set m to 1 for naive distribution and set it to the optimal values (derived by Equation (6)) for partial distribution. The cost is reduced under partial distribution compared to the naive distribution as more bandwidth is saved to improve the access time. Thus, the partial distribution with the theoretical optimal m values performs well.

6.3 Evaluation of index space requirement

Now we evaluate the space requirement of index. Fig. 16 shows percentage of the space allocated for one broadcast cycle, which is used for index dissemination, i.e., $\frac{index}{broadcast\ round}$ under different m values. When $m = 1$, partial distribution degrades to naive distribution, and roughly 11% of the bandwidth is actually for index. However, with m increasing, the percentage decreases sharply first, and remains around 2.3% when m is larger than 10.

6.4 Evaluation of Index Construction Cost

Our last set of experiments is to evaluate the index construction cost. In general, it involves following four tasks, 1) constructing Non-deterministic Finite Automaton (NFA) with submitted queries; 2) Generating documents broadcasting sequence by scheduling algorithms according to the pending queries at the moment; 3) Constructing the compact index based on scheduled documents; 4) Use the constructed NFA and Compact Index to build Pruned Compact Index. Notice that the construction of NFA is necessary for us to use YFilter to find all the matched documents for each submitted query. We can see that Task 1 and Task 2 can be performed paralleled, Task 2 and Task 3 are serial, and Task 4 is serial with Task 1 and Task 3 respectively.

In our implementation, NFA is built incrementally and we update NFA whenever a new query is submitted to the server. Consequently, the time cost of building NFA incrementally during a cycle can be neglected. Task 2 depends on the scheduling algorithm which is not the focus of our work. For Task 3, we build the compact index in advance as it is only dependent on the whole XML document collection that remains unchanged during broadcast. In other words, the time cost of Task 4 actually dominates the index construction cost.

The experimental study result is reported in Fig.17. Obviously, the construction time grows with the increasing queries and arrives almost 900ms when the number of queries reaches 100,000. However, compared to the fact that dissemination of an XML document in NITF dataset takes more than 100ms in a wireless broadcast channel with 1Mbps, the index construction time is acceptable. In addition, the index construction can be further improved via distributed computing.

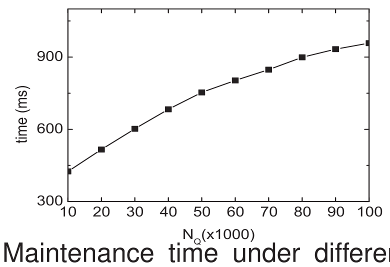


Fig. 17. Maintenance time under different number of queries

7 Conclusion

In this paper, we present a wireless on-demand broadcast method to support XPath queries and provide XML document retrieval. First of all, a two-tier index structure based on DataGuides is proposed. We develop a pruning technology to reduce the index size via removing all the nodes that are not requested by any client, and design a two-tier structure that can further cut down the index size and facilitate the index distribution. In addition, two distribution strategies, namely naive distribution and partial distribution, are presented to interleave the index information with XML documents in the wireless channels efficiently. We have conducted a comprehensive set of experiments to evaluate the performance of proposed index structure as well as the index distribution strategies. In the near future, we plan to study the impact of user query patterns on the system performance and develop a prototype to demonstrate the practical power of our proposed two-tier indexing method.

References

[1] K. Candan, W. Hsiung, S Chen, J. Tatemura, D. Agrawal, "AFilter: Adaptable XML Filtering with Prefix-Caching Suffix-Clustering," *VLDB'06*, pp. 559-570, 2006.
[2] Y. Diao, M. Altinel, M. Franklin, H. Zhang, P. Fischer, "Path Sharing and Predicate Evaluation for High-Performance XML Filtering," *TODS*, vol. 28, no. 4, pp. 467-516, 2003.
[3] R. Goldman, and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," *VLDB'97*, pp.436-445, 1997.
[4] N. Prabhu and V. Kumar, "Data Scheduling for Multi-item and Transactional Requests in On-demand Broadcast," *MDM'05*, pp. 48-56, 2005.
[5] G. Lee, S. Lo, "Broadcast Data Allocation for Efficient Access of Multiple Data Items in Mobile Environments," *MONET*, vol. 8, no.4, pp. 365-375, 2003.
[6] J. Xu, D. L. Lee, Q. Hu, and W.-C. Lee, "Data Broadcast," *Handbook of Wireless Networks and Mobile Computing*, Chapter 11, ISBN 0-471-41902-8, pp. 243-265, 2002.
[7] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on Air: Organization and Access," *TKDE*, vol. 9, no.3, pp. 353-372, 1997.
[8] Tova Milo, Dan Suciu, "Index Structures for Path Expressions," *ICDT'99*, pp. 277-295, 1999.
[9] Raghav Kaushik, Pradeep Shenoy, Philip Bohannon, Ehud Gudes, "Exploiting Local Similarity for Indexing Paths in Graph-Structured Data," *ICDE'02*, pp. 129-140, 2002.
[10] Chen Qun, Andrew Lim, Kian Win Ong, "D(k)-Index: An Adaptive Structural Summary for Graph-Structured Data," *SIGMOD'03*, pp. 134-144, 2003.
[11] Raghav Kaushik, Philip Bohannon, Jeffrey F. Naughton, Henry F. Korth, "Covering indexes for branching path queries," *SIGMOD'02*, pp. 133-144, 2002.
[12] Wei Wang, Hongzhi Wang, Hongjun Lu, Haifeng Jiang, Xuemin Lin, Jianzhong Li, "Efficient Processing of XML Path Queries Using the Disk-based F&B Index," *VLDB'06*, pp. 145-156, 2005.
[13] Weiwei Sun, Ping Yu, Yongrui Qin, Zhuoyao Zhang, Baihua Zheng: Two-Tier Air Indexing for On-Demand XML Data Broadcast. *ICDCS'09*, pp. 199-206, 2009.
[14] S. Park, J. Choi, S. Lee, "An Effective, Efficient XML Data Broadcasting Method in a Mobile Wireless Network," *DEXA'06*, pp. 358-367, 2006.
[15] J. P. Park, C. -S. Park, and Y. D. Chung, "Energy and Latency Efficient Access of Wireless XML Stream," *JDM*, vol. 21, no. 1, pp. 58-79, 2010.
[16] Chang-Sup Park, Chung Soo Kim and Yon Dohn Chung, "Efficient Stream Organization for Wireless Broadcasting of XML Data," *ACCOMS'05*, pp. 223-235, 2005.
[17] Jun Pyo Park, Chang-Sup Park and Yon Dohn Chung, "Lineage Encoding: An Efficient Wireless XML Streaming Supporting Twig Pattern Queries," *TKDE*, 2011.
[18] Z. Vagena, M. Moro, V. Tsotras, "RoXSum: Leveraging Data Aggregation and Batch Processing for XML Routing," *ICDE'07*, pp. 1466-1470, 2007.
[19] A. Diaz, and D. Lovell, *XML Generator,* http://www.alphaworks.ibm.com/tech/xmlgenerator, 1999.
[20] Weiwei Sun, Zhuoyao Zhang, Ping Yu, Yongrui Qin, "Efficient data scheduling for multi-item queries in on-demand broadcast," *EUC'08*, pp. 17-20, 2008.

Weiwei Sun received his bachelor's degree in Computer Science and Technology at Fudan University in June 1992. From Sept.1995 to July 1998 and Sept.1998 to Dec.2001, he received his master and Ph.D's degree respectively in Computer Software and Theory at Fudan University. Now he is an associate professor of School of Computer Science and the Director of Mobile Data Management Laboratory of Fudan University. His interests include wireless data broadcast, spatial database, service computing in ad hoc and etc.

Yongrui Qin received the bachelor's degree and the master's degree in Computer Science and Technology at Fudan University in 2005 and in 2008. He also worked as Assistant Software Engineer and Research Assistant before Ph.D study. He is currently a Ph.D student in School of Computer Science, the University of Adelaide. His research interests include internet of things, pervasive computing and mobile data management.

Jingjing Wu received the bachelor's degree in Computer Science and Technology at Fudan University in 2009, After that, she went on working toward the master's degree in Computer Science and Theory at Fudan University. Her research interests include wireless data broadcast and XML data broadcast.

Baihua Zheng received the bachelor's degree from Zhejiang University and the Ph.D's degree in the Department of Computer Science, Hong Kong University of Science and Technology. She currently is an associate professor of the School of Information Systems, Singapore Management University. Her research interests include mobile/pervasive computing and spatial database.

Zhuoyao Zhang received the bachelor's degree and master's degree in Computer Science and Technology at Fudan University in 2006 and in 2009. She is pursuing a Ph.D's degree in the Department of Computer and Information Science at the University of Pennsylvania. Her research interests include distributed resource management, data sharing in distributed system, mobile computing and ad hoc networks.

Ping Yu received the bachelor's degree of Computer Software from Huazhong University of Science and Technology in 1993, then she got her master's degree of Computer Software in Sun Yat-Sen University in 1996. In 2005, she started to work for the Ph.D's degree of Computer Software and Theory in Fudan University and got her Ph.D's degree in 2008. Her interests include mobile data management, mobile computing and database.

Peng Liu received the bachelor's degree in Computer Science and Technology at Fudan University in 2008, and now he is working for his master's degree in school of Computer Science and Technology at Fudan University. His research interests include wireless data broadcast and XML data broadcast.

Jian Zhang received the bachelor's degree in Computer Science and Technology at Sun Yat-Sen University in June 2011. Now he is working for his master's degree in School of Computer Science and Technology at Fudan University. His research interests include wireless data broadcast and XML data broadcast.