In-broker Access Control for Information Brokerage Systems

Fengjun Li, Bo Luo, Peng Liu, Dongwon Lee, Prasenjit Mitra, Wang-Chien Lee, and Chao-Hsien Chu
The Pennsylvania State University, University Park

***Abstract*—An XML brokerage system is a distributed XML database system that comprises data sources and brokers which, respectively, hold XML documents and document distribution information. Databases can be queried through brokers with no schema-relevant or geographical difference being noticed. However, all existing information brokerage systems view or handle query brokering and access control as two orthogonal issues: query brokering is a system issue that concerns costs and performance, while access control is a security issue that concerns information confidentiality. As a result, access control deployment strategies (in terms of where and when to do access control) and the impact of such strategies on end-to-end system performance are neglected by existing information brokerage systems. In addition, data source side access control deployment is taken-for-granted as the "right" thing to do. In this paper, we challenge this traditional, taken-for-granted access control deployment methodology, and we show that query brokering and access control are not two orthogonal issues because access control deployment strategies can have significant impact on the "whole" system's end-to-end performance. We propose the first in-broker access control deployment strategy where access control is "pushed" from the boundary into the "heart" of the information brokerage system. We design and evaluate the in-broker access control scheme for information brokerage systems. Our experimental results indicate that information brokerage system builders should treat access control as a system issue as well.**

***Index Terms*—Information Brokerage System, Role Based Access Control, XML**

I. Introduction

Information sharing is becoming increasingly important in recent years, not only among organizations with common or complementary interests, but also within large organizations and enterprise that are becoming ever more globalized and distributed. Multiple divisions cooperate within large multi-national enterprise as well. For example, in GM, to maintain a proper stock level of parts, people in supply management division need to check the sale information (of car models) gathered and managed by sales people world-wide. In such information sharing systems, the data gathered by a specific division are typically stored and maintained in a database *local* to the division, but the needs to access the data may potentially come from any *remote* division.

Although the Internet and various virtual private networks provide good data communication links, there are major challenges in (a) achieving scalable, agile and secure remote access of distributed data; (b) handling the heterogeneity among data management systems and data formats which are not always structured and may be incompatible with each other; (c) handling the dynamics of modern business applications (where new schema elements may emerge everyday); and (d) location discovery. Classic distributed database cannot meet the challenges, for example, they require a static fully structured "global" database schema that may not exist in most cases. While the globalization has brought intrinsic and increasing needs to data sharing, access to presents complications such as query brokering under heterogeneity, location discovery and secure information access.

To tackle these challenges, mediation and federation based information brokering technologies have been proposed. In particular, recent eXtensible Markup Language (XML) has become a promising solution [1] by integrating incompatible data while preserving semantics. An XML-based information brokerage system comprises data sources and brokers which, respectively, hold XML documents and document distribution information. In such systems, databases can be queried through brokers with no schema-relevant or geographical difference being noticed.

However, from the security, especially access control, point of view, existing information brokerage systems have a fundamental misconception. That is, they view or handle query brokering and access control as two orthogonal issues: query brokering is a system issue that concerns costs and performance, while access control is a security issue that concerns data confidentiality. As a result, access control deployment strategies (in terms of where and when to do access control) and the impact of such strategies on end-to-end system performance are neglected by existing systems. In addition, data source side access control deployment is taken-for-granted as the "right" thing to do. In this paper, we challenge this traditional, taken-for-granted access control deployment methodology, and show that query brokering and access control are **not** two orthogonal issues because access control deployment strategies can have significant impact on the "whole" system's end-to-end performance.

Our **contributions** are: (1) we propose the first in-broker access control deployment strategy where access control is "pushed" to the brokers; (2) we design three specific in-broker approaches to implement the "pushing" idea; (3) experiments are taken to show that in-broker access control can significantly improve the performance of memory consumption, end-to-end query directing time and network occupancy without

Email addresses: {fengjun, bluo, pxl20, dongwon, pum10, wul2, chc4}@psu.edu

hurting the system-wide security[1].

The rest of this paper is organized as follows. In Section IV, we examine the existing architecture of brokerage systems and propose a new one called in-broker architecture. In Section III, we give a brief description on access control model and one specific enforcement called QFilter. In Section V, we extend QFilter and propose three new techniques which combine data locating and access control inside brokers. In Section VI, we do performance evaluation and show that our proposal is better than existing ones in virous aspects. Related work and conclusion are given in Sections II and VIII respectively.

II. Related Work

Publish/Subscribe systems (e.g., [3], [4]) are event-based and provide many-to-many communication between event publishers and subscribers. On the other hand, on demand information distribution systems deliver information upon user queries. Our approach is an information distribution system for its spontaneous query answering capability. As an XML-based overlay network, [5] proposed a mesh-based overlay network that supports XML queries. In [6] XML content-based routing is addressed using the query aggregation scheme given in [7]. In [8], content-based routing of XPath queries in P2P systems is studied. However, none of these work addresses the integration of information brokerage and access control, which is one of our main emphases. The Content Distribution Networks (CDN) provide an infrastructure that delivers static or dynamic Web objects to clients from cache or replicas to off-load the main site [6], [9]. This differs from our approach in that it does not give users a powerful query language. Also, our focus is how to distributes access controls, not data, among brokers. [10] gives a good overview on access control in collaborative systems. Although many, existing "distributed" access control theories and techniques focus on the policy, modeling, and flexibility aspects. However, our work focuses on performance-optimizing enforcement strategies using in-broker access controls.

We adopt the access control model proposed by [11], [12] in the brokerage system. However, our system is not tightly coupled with one specific model, and the proposed techniques can be applied to other AC models (e.g., [13], [14], [15], [16]). As to enforcing XML access controls, by and large, existing approaches either use "views" (e.g., compressed accessibility map of [17]) or rely on the underlying XML engine (e.g., [18]). Our proposal is based on the QFilter [19] – query re-writing access controls – that does not use views nor require any support from XML databases. Finally, compared with various researches on the equivalence/containment/re-writing of XML queries [20], our approach is NFA-based and security-driven. In this paper, we extend the idea of QFilter further to the context of in-broker access controls. Therefore, our access controls can occur anywhere in the network freely – at client, server, and in-between.

[1]An earlier version of this paper has been published in [2]

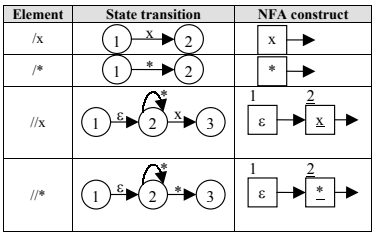


Fig. 1. Building blocks for NFA construction

III. Background

A. XML Access Control Model

In this paper, we adopt the fine-grained XML access control model similar to [11], and incorporate Role Based Access Control [12]. In our model, administrators assign *roles* to users. Each role is given a set of access rights to data (XML nodes). The authorization is specified via 5-tuple access control rules (ACR): R = {*subject, object, action, sign, type*}, where (1) *subject* is to whom an authorization is granted (i.e., role); (2) *object* is a set of XML nodes specified by XPath; (3) *action* is one of "read," "write," and "update"; (4) *sign* $\in \{+, -\}$ refers to access "granted" or "denied," respectively; and (5) *type* $\in \{LC, RC\}$ refers to either "Local Check" (i.e., authorization is only applied to attributes or textual data of context nodes), or "Recursive Check" (i.e., authorization is applied to context nodes and propagated to all descendants). By default, access is denied to all nodes whose authorizations are not specified. When a node has more than one relevant rule, the negative rules take precedence upon positive rules.

B. Introduction to QFilter

One of the recent developments of XML access control is to enforce access control on input queries [21], [19], [22], [23], [24]. In this section, we introduce a state-of-the-art technique, called QFilter [19], that we recently proposed. QFilter captures a set of access control rules using a Non-deterministic Finite Automata (NFA), and re-writes parts of incoming query Q that violate the access rights, to yield a safe query Q'.

1) QFilter Construction: QFilter construction process is very similar to regular NFA construction. It takes XPath expressions from access control rules to build NFA. We accept a subset of XPath – parent-child (/x), ancestor-descendant (//x), and wildcard (/*, //*). Figure 1 illustrates how NFA state transitions are constructed from these four basic XPath steps. The NFA for a complete set of ACR is formed by linking the states in sequence. Let us use ACR of Figure 2 as an example (to simplify the presentation, let us focus only the *object* part of ACR, ignoring the rest). The QFilter construction starts from R_1: we create states 0 and 1 for XPath step `/site`; then a transition on token `categories` is created to state 3 for XPath step `/categories`; and so on. Finally, the constructed QFilter is shown in Figure 2(b).

2) QFilter Execution: In the context of access controls, QFilter captures ACR^+ and ACR^-. For an input query Q, QFilter has three types of operations: (1) **Accept**: If answers of Q are contained by ACR^+ (i.e., Q asks for answers granted

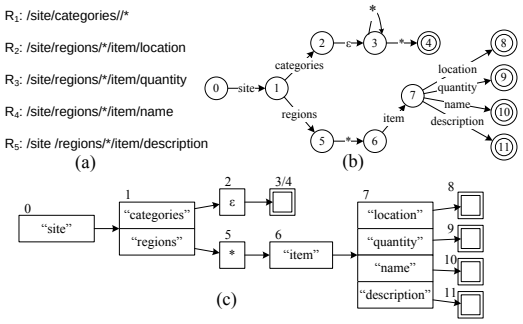
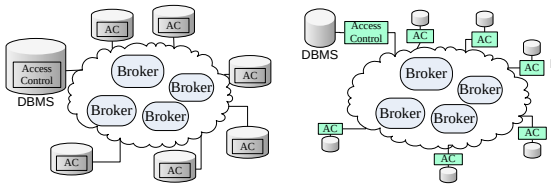


Fig. 2. An example to construct a QFilter: (a) Sample ACRs (*object* part only); (b) State transition map; (c) NFA transition.



(a) Embedded Access Controls(EAC) (b) Source-side Access Controls(SAC)

Fig. 3. Three architectures of information brokerage systems.

by ACR^+) and disjoint from ACR^- (i.e., Q does not ask for answers blocked by ACR^-), then QFilter accepts the query as is: $Q' = Q$; (2) **Deny**: If answers of Q are disjoint from ACR^+ or contained by ACR^-, QFilter rejects the query outright: $Q' = \emptyset$; (3) **Rewrite**: if only a partial answer is granted by ACR^+ or blocked by ACR^-, QFilter rewrites Q into the ACR-obeying output Q'. Finally, Q' is guaranteed to be: (i) contained in Q, (ii) contained in ACR^+ and (iii) disjoint with ACR^-. Note that, for rewritten queries, the output could be "UNION" of several XPath queries (To be more strict, "DEEP UNION" should be used [25]).

QFilter executes a filtering process for the input query Q and returns the safe one Q' as output. The filtering operations are accept, deny and rewrite where Q' equals to Q, $\{\}$, and $Q \cap ACR$ respectively. For deterministic transitions, QFilter works the same way as a regular NFA does – an incoming query is either "accepted" or "denied" by the automaton. The execution of the QFilter gets more complicated when non-deterministic operators ("*" and "//") and predicates occur in either input query Q or ACR. For instance, if Q contains a wildcard "*" but NFA only accepts a certain token "x", the wildcard is accepted and rewritten into "x". Due to space constraints, we will refer the details to [19].

Instead of simply filtering out queries that do not satisfy access control policy as in [21], QFilter takes extra steps to rewrite queries based on AC policy and passes revised queries to underlying DBMS. In doing so, QFilter not only achieves security for (almost) free, but also enjoys a faster query evaluation time through query re-writing.

For instance, if we only have ACR^+: {user, /site/regions/*/item/name, +, read, LC} and {user, /site/regions/*/item/location, +, read, LC} (*user* can only read `name` and `location` nodes under `item`). For the input query "$//item/*$", the QFilter would yield the following re-written query "$/site/regions/*/item/name$ UNION $/site/regions/*/item/location$".

IV. Information Brokerage System Architecture

Consider an *information brokerage system* where sensitive information is shared among geographically distributed participants (e.g., users and data sources). In general information brokering process, XML queries created by a user are forwarded to data sources by intermediate brokers. Since multiple data sources may be relevant to one XML query, replies from all relevant data sources will be merged to provide an aggregate view to the user.In this process, brokers perform as a bridge connecting users and data sources, so they are necessary to know who holds the required data and where they are located. To make the exposition simple, we assume that each broker has a full knowledge of whereabouts of stored data. Therefore, each broker may direct an inquiry to relevant data sources without consulting others (i.e., single-hop brokering). Since query brokering is not the focus of this paper, we will limit our investigation to the case of single-hop brokering. Nevertheless, supporting multi-hop brokering is part of our future work.

Besides query brokering, access control is another important issue in information brokering. Accessibility of the requested data are based on access control policy. However, most existing information brokerage systems handle query brokering and access control as two orthogonal issues and adopt the take-for-granted access control deployment where access control enforcement is embedded in DBMS. To challenge this traditional viewpoint, we describe two alternative architectures in which access control enforcement is pulled out of DBMS.

A. Embedded Access Control Architecture (EAC)

In the traditional brokerage systems, the job of security enforcement is solely left upon the shoulder of DBMS. For instance, administrators define access controls inside DBMS; any query needs to pass access controls before it is processed. In a sense, the enforcement of access controls is "embedded" into DBMS. Figure 3(a) illustrates this architecture, named as *Embedded Access Controls (EAC)*.

Since this security enforcement is highly depended on DBMS, it puts much burden in choosing a proper DBMS. Considering our large-scale, heterogenous data source assumption, it is not an appropriate solution. Moreover, most embedded access control adopt the view-based enforcement mechanism, to the best of our knowledge. Logical or physical views are created over the roles, and each role is granted full access to his own view. However, view construction consumes large storage space, especially when the number of roles scales.

B. Source-Side Access Control Architecture (SAC)

Since XML query, data and access control enforcement are independent building blocks in an XML database system, it is possible to pull access control enforcement out of DBMS. For instance, in [19], we show that access controls can be supported via query rewriting outside of DBMS, thereby decoupling the tie between access controls and DBMS. One of

the many benefits of this architecture is that access controls can be enforced without the support from underlying DBMS. For instance, none of the commercially available XML databases is capable of supporting access controls. Figure 3(b) illustrates this architecture, named as *Source-side Access Controls (SAC)*.

[19] claimed a superior performance compared with embedded access control approaches. However, access control mechanism is still deployed at data source, which is the boundary of information brokerage systems. Queries without right accessibility are still routed through the system and get denied at data source, resulting in poor performance and network resources waste.

C. In-Broker Access Control Architecture (IAC)

Intrigued by the SAC scenario, we pull the access controls further to the brokers: from the boundary into the "heart" of information brokerage systems. In this way, security check is enforced when users access the network. Figure 4 illustrates this architecture, namely *In-broker Access Controls (IAC)*.

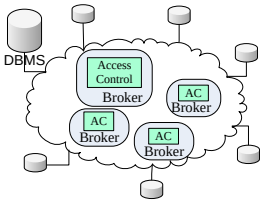


Fig. 4. In-broker Access Controls(IAC) information brokerage architecture

We claim that query brokering and access controls are *not* orthogonal issues. By integrating them properly, the whole system benefits from this integrated design and end-to-end performance improves without hurting system-wide security. For instance, a query that cannot access its requested data will be rejected at very early steps at the brokers and never have a chance to reach any data source. In one way, overall processing time is dramatically shortened and network communication is greatly lessened; in another way, system-wide security benefits from the early denial of potential suspicious actions. Early denial of the queries not only shortens the end-to-end query response time of the initiating user in average, but also save the network resources by not dispatching them to the oriented data sources. The system-wide security benefits from the early denial of suspicious actions and intrinsic replication among brokers. In our in-broker architecture, suspicious actions are denied in the middle (at brokers) instead of at the far-end (at DBMS), thus avoiding potential misfeasance. For instance, a lower-level user or even an unauthorized user could send a kind of "snooping query" to trace the denial from sensitive databases or even explore the distribution of the whole system under the traditional architecture. On the other hand, replication is far easier at the broker level, for only access control and location information needs to be replicated instead of the whole databases. A good number of replicated brokers, and possible dual access control (i.e., double-check or validate if an access control policy is correctly enforced at the data source side) help detecting and recovering from malicious attacks such as DoS. The only concern of pushing access control to the brokers is their trust level. Since our brokerage system is used in intra-organizations and inter-organizations, we assume certain level of trust in brokers. In more severe cases, we can use dual access control and ene-to-end auditing systems to help monitoring brokers' behavior. We will discuss this in more detail in Section VII.

V. Approaches for In-broker Access Control

In this section, we introduce new index and access control approaches, the brokering indexer and the Multi-role QFilter, and show how we combine both to create an Indexed Multi-role QFilter. In the previous sections, we analyze the complications of the large-scale distributed nature and these approaches fulfil the requirements well.

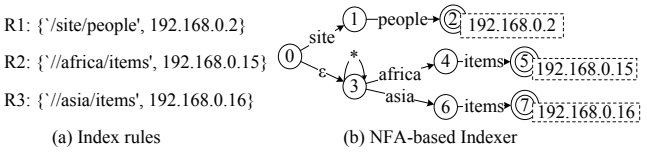
A. Brokering Indexer

In XML brokerage systems, users send queries without knowing the data location. Brokers have physical distribution information of XML documents. In our setting, a query is routed using single-hop brokering, i.e., any broker is able to determine the data location of any query. Note that multi-hop routing might be used in lower layers, e.g., if the destination is identified by its IP, IP layer routing is multi-hop.

An index rule is described as R^{ind}=$\{obj,\ des(s)\}$ where "*des*" is a network address, and "*obj*" is an XPath expression, as shown in Figure 5(a). The index table look-up is essentially one-to-many XPath matching. We design a QFilter-like NFA (Indexer) to handle it. As shown in Figure 5, the Indexer is constructed with XPath expressions from index rules ($R^{ind}.obj$). At each *accept state*, $R^{ind}.des$ is attached. Destination lookup is like any NFA execution, which is to match user queries with routing rules captured in the Indexer. During execution, $R^{ind}.des$ is attached to the query when appropriate. Finally, accepted queries are forwarded to the list of destinations attached to it. When a query does not match any index rule, it means no known data source has the requested data, thus the query is dropped. A query could also match several accept states, thus all the destinations are attached. E.g. query "`//items`" sent to Indexer shown in Figure 5 matches accept states 5 and 7, thus will be forwarded to both destinations. In our approach, users take the responsibility of joining answers from different data sources.

Since the indexing table look-up process is essentially one-to-many XPath matching, we designed a QFilter-like extended NFA to handle the problem. Our system constructs an NFA Query Indexer using the XPath expressions of the indexing rules in the same way as a QFilter. At each accept state, the destination address is attached. The execution is very similar to QFilter execution (Query filtering). However, the destination information is processed in the following cases: (1) when a query passes an accept state while brokering in the NFA, the destination address stored at that state is attached to the query; (2) if a query stops at a middle or accept state of the NFA, the state (if it is an accept state) and all its descendant accept states are traversed and all the stored destination states are attached to the query. The query is accepted; (3) if a query is routed to an accept state and the incoming token (element

name of next step) is not found in the state transition table, the destinations that are stored at the current accept state is attached to the query. The query is accepted; and (4) if a query is routed to a non-accept state and the incoming token is not found in the state transition table, the query will be rejected since no information on the location of the requested content is available. An accepted query is forwarded to the list of destination addresses attached to the query.



(a) Index rules (b) NFA-based Indexer

Fig. 5. An example of NFA-based Indexer

B. The Multi-Role QFilter Approach

1) QFilter Array (QA): The QFilter approach described earlier is designed for single role. In a network setting, access control for multiple roles with individual *ACR* is needed. To address this need, a straightforward extension to QFilter is to use an array of QFilters (called *QFilter Array*), where each QFilter is constructed, stored and executed independently. When a query is submitted, the role of user is identified and the corresponding QFilter is located from the array to process the query.

One serious drawback of QFilter Array approach is that its memory usage grows linearly with the number of roles in the system. When large number of roles exists, it soon grow beyond size of main memory, therefore, the system performance dramatically degrades . To tackle this problem, we introduce Multi-Role QFilter.

2) Multi-Role QFilter (MRQ): We observe that access control rule sets for different roles are often similar, therefore their QFilters are also similar. The idea of *Multi-Role QFilter* (MRQ) is to merge similar QFilters into one uniform data structure instead of storing them in an array. Since each QFilter is constructed for one particular role, this merging method should identify access control rules to the roles. In our design, we use an Boolean array (*bitmap*) for its constant lookup cost.

MRQ Construction. We construct MRQ by annotating each QFilter state with two bitmaps: *access_list* and *accept_list*, where each bit represents a Boolean value for a role. Thus a corresponding pair *(access_value, accept_value)* is assigned to each role. The *access_value* indicates whether the role has access right to this state and the *accept_value* indicates whether the state is an accept state for this role. Figure 6 shows an example: there are two roles with individual ACR; a QFilter Array consisting of two individual QFilters is shown first, and the MRQ that serves both roles is shown underneath. The MRQ (labeled "Merged" in Figure 6) contains two bitmaps at each state to indicate the accessibility of each role, e.g. the first three states are accessible by both roles (the *access_value*s are 1) but no one is an accept state (the *accept_value*s are 0). State 3 is the accept state for role 1 only, while 4 is accessible and accept state for role 2 only.

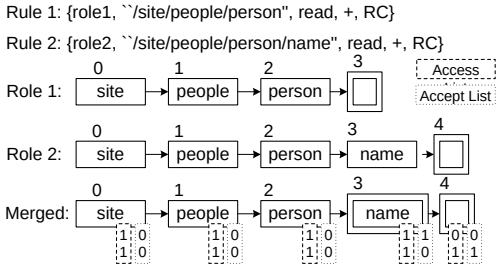
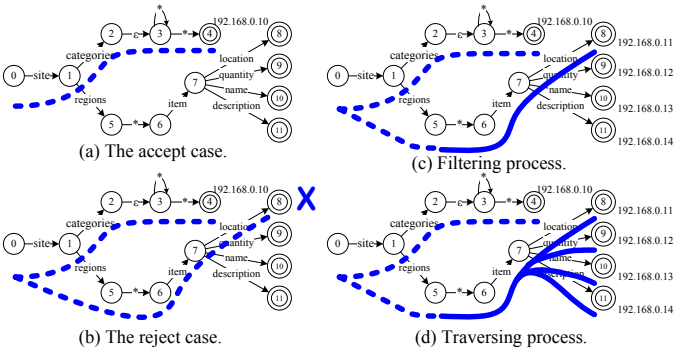


Fig. 6. Merge QFilters to Multi-role QFilter.

MRQ Execution. Similar to a single QFilter, for an input query $\{Q, role\_id\}$, MRQ has three types of outputs: **Accept**, **Deny**, or **Rewrite**. During the execution, at each MRQ state, the access right of the role is first checked with *access_list* based on the $role\_id$. Only when the *access_value* is 1, the execution proceeds to subsequent states. In this manner, the *access_value* restricts the region, in which a query may traverse in a MRQ.

C. Indexed Multi-Role QFilter Approach

1) Implementation: In above approaches, access control enforcement is moved from data sources to the center of the network – the brokers. Therefore, brokers hold both indexing and access control mechanisms. When user query Q is submitted, MRQ processes it to safe query Q', then Indexer locates the data source. Since two mechanisms with similar structure reside at the same place, it is natural to merge them to improve efficiency. Therefore, we propose "Indexed Multi-Role QFilter" (*IMQ*), which captures both indexing and access control rules in one NFA structure. A query Q sent to IMQ yields the output of $\{Q', \{des(s)\}\}$, where Q' is the safe query.



(a) The accept case. (c) Filtering process.

(b) The reject case. (d) Traversing process.

Fig. 7. IMQ Construction.

IMQ Construction. As described in Section V-A, an index rule (IR) is: R^{index}=$\{obj, des\}$. The IMQ construction consists of three steps: (1) **MRQ Construct**: construct a MRQ using ACR; (2) **IR Filtering**: executing the XPath of $R^{index}.obj$ in MRQ; and (3) **Attaching**: attach $R^{index}.des$ to the current and descendent accept states.

Instead of giving the exhausted algorithm, we use an example to show IMQ construction. As shown in Figure 7 (we assume that MRQ is already constructed):

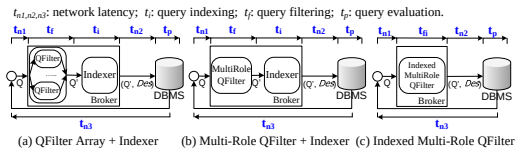


Fig. 8. Three brokering approaches, where $t_{n1,n2,n3}$, t_i, t_f and t_p represent time for network latency, query indexing, filtering and processing respectively.

(a) IR {/site/categories//*, 192.168.0.10} reaches state 4, indicating "categories" nodes are accessible and are located at 192.168.0.10; (b) IR {/site/regions/item/payment, 192.168.0.5} does not reach any accept state, indicating no user is allowed to access the nodes although they are located at 192.168.0.5 (as a result, this rule is not included in the final IMQ); (c) and (d): XPath of IR {/site/regions, 192.169.0.13} stops at state 5, then we attach its destination to all the descendant accept states, i.e., 8, 9, 10 and 11. This means, 192.169.0.13 holds "regions" nodes, but only some descendants are accessible according to ACR.

IMQ Execution. For a user query $(Q, role\_id)$, IMQ execution is almost identical to MRQ execution, except that the appropriate des addresses are attached to accept and rewritten queries. Finally those queries are forwarded to all the attached destinations.

VI. Experimental Validation

We have implemented the three brokering approaches (shown in Figure 8) proposed in Section V. In this section, we present experiments based on this implementation. In the first experiment, we investigate how memory and query filtering time change with parameters (the number of roles and the number of rules per role) in QA and MRQ approaches. A reasonable setting is then chosen for the second experiment, where we measure the memory consumption and the end-to-end query brokering time for MRQ and IMQ approaches and show the IMQ approach performs best.

A. Query filtering

Settings. We use the well-known XML benchmark XMark [26] DTD. It defines 77 elements and 16 attributes for an on-line auction scenario. In rule XPath generation, maximal depth of the XPath expressions is set to 6 [27]. Synthetic rules are randomly generated: (1) with 10% wildcard (* or //) probability at each step, and (2) without wildcard. Then we vary *number of roles* from 10 to 500, and *number of rules per role* from 5 to 300. To evaluate the query filtering time, we generate 500 synthetic queries, each with one predicate, and 10% wildcard probability at each step.

Memory Cost. Memory consumption of QA approach and MRQ approach is shown in Figure 9: rules in Figure 9(a) and (b) has wildcards, while rules in (c) and (d) has no wildcard. As expected, memory consumption in QA approach is proportion to the *number of roles*, which is same as number of QFilters in the Array. But memory usage increases below-linear with the *number of rules per role*, since rules in the same QFilter shares NFA states. Especially, when there are more rules for each role, there is higher possibility for states

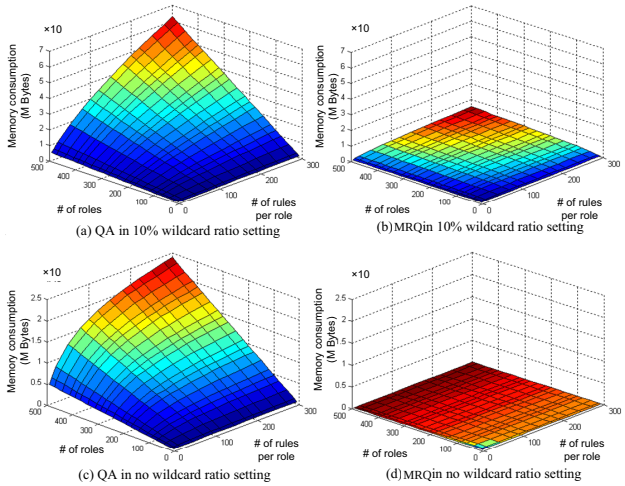


Fig. 9. Memory usage of QA and MRQ.

sharing. In Figure 9(b), we can see a significant saving in memory comparing with (a). Because, in MRQ, all rules are contained in a big QFilter-structure, rules from different roles are able to share NFA states. Next, rules with no wildcard is used, with experiment results shown in Figure 9 (c) and (d). Under this setting, only 105 distinct XPath expression are generated for ACRs, and the percentage of state sharing is extremely high. In both settings, the memory usage for MRQ is one order of magnitude smaller than that of QA. Similar result is obtained for the ACR set with predicates. We do not list the result here due to space constrains.

Query Filtering Time There are three possible results when queries are processed, in QA or MRQ: denied, accepted, or rewritten. Accepted queries take more time to be processed, while rewritten queries take the longest. In our experiments, 198 queries are rejected and 302 queries are accepted/rewritten.

Figure 10(a) and (b) show the average query processing (filtering) time for all queries. It is clear that MRQ approach is about four times slower than the QA approach for all queries.

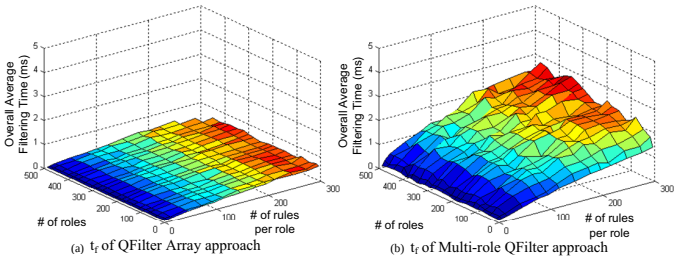


Fig. 10. (a) and (b) show the absolute filtering time under two approaches respectively.

Analysis. There is a tradeoff between memory consumption and query filtering time. The query filtering time is in the scale of milliseconds, much smaller than the network latency (in the scale of hundreds of milliseconds). Thus, its impact is not as significant as memory consumption, which is a major concern for brokers. We conclude that Multi-Role QFilter is a better solution than QFilter Array.

TABLE I
COMPARE THE MEMORY AND QUERY BROKERING TIME OF THREE IN-BROKER APPROACHES.

Approaches (with 1000 indexes)	QA+I	MRQ+I	IMQ
Memory for Index (KB)	418	418	-
Memory for access control (KB)	2934	969	-
Memory for in-broker total (KB)	3352	1387	1094
Time for Index (ms)	402	1131	-
Time for access control (ms)	105	482	-
Time for in-broker total (ms)	507	884	447
Time for in-broker average (ms)	1.014	1.768	0.895
Approaches (with 4000 indexes)	QA+I	MRQ+I	IMQ
Memory for Index (KB)	1027	1027	-
Memory for access control (KB)	2934	969	-
Memory for in-broker total (KB)	3961	1996	1119
Time for Index (ms)	1131	1131	-
Time for access control (ms)	105	482	-
Time for in-broker total (ms)	1638	2015	459.3
Time for in-broker average (ms)	3.276	4.030	0.919

B. Query filtering and indexing

In this experiment, we compare the three in-broker access control approaches, as shown in Figure 8: QA+Indexer, MRQ+Indexer, and IMQ (Indexed MRQ).

Settings. We fix the *number of roles* to 80 and the *number of rules per role* to 50, and randomly generate synthetic access control rules, with 10% wildcard probability at each XPath step and one predicate for each rule. We also generate synthetic XPath expressions for indexing rules at 10% wildcard probability at each step. Since predicate parsing in indexing is not supported in our index scheme, the index paths are generated without predicate. Two sets of indexing rules are built: (1) with 1000 indexing rules (S_{P1}), and (2) with 4000 indexing rules (S_{P2}). The same synthetic query set as in the first experiment is used. Since access control rules, index paths and the queries are all randomly generated synthetic rules, which offset the impact of rule pattern, there is no need to repeat the experiments at the same setting. We do take a set of experiments which result in similar outputs. In the following discussion, we only list the result of one experiment.

Memory Cost. Memory cost for brokering includes the consumption for both access control and indexing. The indexer with 1000 and 4000 index paths consumes about 418KB and 1027KB memory respectively. Overall memory consumption of three mechanisms is summarized in Table I. It is clear that IMQ requires the least amount of memory, while MRQ+Indexer consumes much less than the naive QA+Indexer approach. By merging the index with the existing MRQ, IMQ (with 1000 index rules) only requires an additional memory (compare with MRQ) of 125KB instead of the original 418KB used by the Indexer. When the amount of index paths increase to 4000, the saving is more significant.

Query Brokering Time. The brokering time includes query filtering time (t_f) and query index time (t_i). The time for directing all 500 queries though S_{P1} and S_{P2} is 402ms and 1131ms respectively, and the average is 0.804ms and 2.262ms respectively. The overall brokering time in three mechanisms are listed in Table I. Since the Indexer is not as efficient as the security check process, it dominates the overall performance especially when the amount of index paths goes large. The QFilter Array approach is tailed by the Indexer even though it performs fifth times better than the Multi-Role QFilter approach. The Indexed Multi-Role QFilter approach performs best because the index process is embedded into its security check.

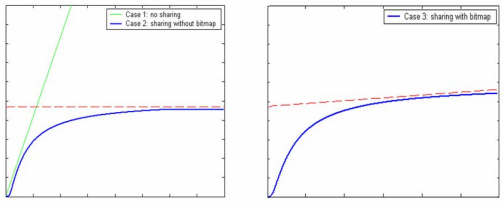


Fig. 11. Analysis of memory usage of MRQ approach

C. Memory usage - revisited

The most distinct feature of Multi-Role QFilter and Indexed Multi-Role QFilter is that they merge QFilters of different roles in order to save memory. Here we give some more insights about the memory usage of MRQ.

For a single QFilter, the required memory is determined mainly by the number of NFA states, which is decided by access control policies. In general, each XPath step generates one NFA state and each descendant-or-self (//) step generates two NFA states. Identical XPath fragments share NFA states.Moreover, since the schema is given, the number of distinct XPath expressions allowed by the schema is bounded (we do not consider value based predicates at this moment). Therefore, both $PC(ACR)$ and $DS(ACR)$ are bounded. In this way, the schema determines a universal set, which contains all the valid XPath expressions under this schema. We call the superset "rule space" (U). If we enumerate all the XPath expressions in the rule space and construct a QFilter, its the largest QFilter (QF_{Max}) under the given schema.

Now let us consider the multi role scenario. Each role has its own ACR, which takes a subset of XPath expressions from the rule space. However, the total number of distinct rules cannot exceed the rule space, i.e.

$$ACR_1 \cup ACR_2 \cup ... \cup ACR_n \in U$$

If we construct a QFilter for each ACR (QFilter array approach), the memory consumption is linear to number of QFilters. However, if we create one QFilter for all the rules (MRQ approach without access list and accept list), it cannot grow beyond QF_{Max}. In Figure 11 (a), the grey line represents the estimation of memory usage of QFilter Array, the dashed line represents the size of QF_{Max}, and the black curve shows the desired memory usage of MRQ.

However, in MRQ, we have an *access list* and an *accept list* (both implemented with Bitmap) attached to each NFA state. Therefore, the upper bound of MRQ size is $size(QF_{Max}) + size(AccessList) + size(AcceptList)$. The size of the two bitmaps grows linearly with the increase of number of roles. However, comparing with the size of QFilter, $size(AccessList)$ and $size(AcceptList)$ are both very small. In Figure 11 (b), the dashed line represents the upper bound of the MRQ size, i.e. $size(QF_{Max}) + size(AccessList) +$

$size(AcceptList)$, while the black curve shows how the actual size of MRQ grows with number of roles.

As a conclusion, the size of MRQ is upper bounded by the rule space (or QF_{Max}), which is determined by the schema. The above analysis is validated by the experiments (Figure 9).

VII. Architecture Level Analysis

In this section, we analyze the advantage of In-Broker Access Control architecture in both qualitative and quantitative manners. We present our analysis in the aspects of system performance and security.

A. Performance Improvement

In the IAC architecture, if access control rules allow a query to access all or part of the requested content (accepted/rewritten query), the original or rewritten query will be forwarded to the query indexer. Otherwise, if the access control rules rejects the query from accessing any requested content, the query is dropped at the broker and/or a error message is returned to the user. In this way, users get faster response when the query is denied. At the same time, less network resources is consumed by only directing accepted and rewritten queries to the data sources.

Murata et al have conducted experiments to investigate the accepted and denied queries [21]. They show that 40% of the queries are type 'G', where all XPath expressions in the query are always granted; 25% queries are type 'D', where at least one of the XPath expressions is always denied; and 35% of the queries are type '-', where at least one XPath expression in the query should be rewritten. We assume a similar distribution in our analysis accordingly.

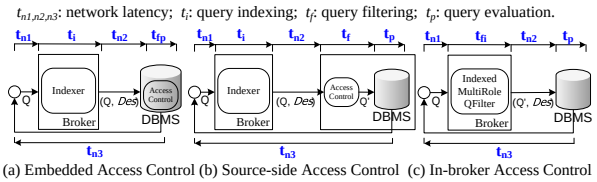
The system performance is analyzed in two aspects: query processing performance (end-to-end query processing time), and network occupancy.

1) End-to-End Query Directing Time.: The efficiency of query processing is evaluated as the time elapsed from user submits the query until he/she receives the answer. As shown in Figure 12, query processing time is calculated as:

- In embedded access control: $t_{n1} + t_i + t_{n2} + t_{fp} + t_{n3}$
- In Source side access control: $t_{n1}+t_i+t_{n2}+t_f+t_p+t_{n3}$
- In In-broker access control: $t_{n1} + t_{fi} + t_{n2} + t_p + t_{n3}$

Note that, different information brokerage architectures differ in the way they forward queries to the data source and conduct security check. The backward answer transmission mechanisms (transfer the answer from data source back to the user) remain the same. Therefore, for the same data, access control policies and query, backward answer transmission time t_{n3} remains constant in all architectures, and we thus focus on query forwarding phase. Moreover, for query processing time, t_p in source side access control and in-broker access control are identical, while t_{fp} in embedded access control is significantly larger because of the node-level security check process. Now, we focus on the query forwarding phase, where all the differences reside.

We define the end-to-end query directing time as the time elapsed from user submits the query until it reaches the database, i.e. the summation of query filtering time (t_f) (except for embedded access control architecture), indexing time (t_i), and network latency (t_n), as shown in Figure 12. General network latency is 200ms [2], thus we assume t_{n1} and t_{n2} for a single query are both 100ms. Since 25% of the queries fail the security check and get rejected at the brokers in IAC architecture, the average time of t_{n2} is reduced to 75ms for IAC. Moreover, for each architecture, we obtain average t_f and t_i via experiments, and summarize the results in Table II. From the results, we can clearly see that the network latency dominates the whole process, and thus the performance under IAC architecture is much better than approaches of EAC or SAC architectures.



(a) Embedded Access Control (b) Source-side Access Control (c) In-broker Access Control

Fig. 12. End-to-end query answering time, where $t_{n1,n2,n3}$, t_i, t_f and t_p represent time for network latency, query indexing, filtering and processing respectively.

TABLE II
End-to-end query forwarding time.

Approach/Time(ms)	t_{n1}	t_f+t_i	t_{n2}	overall
EAC	100	<1	100	$\sim$ 200
SAC with QA	100	1.014	100	201.014
SAC with MRQ	100	1.768	100	201.768
IAC with QA	100	1.014	75	176.014
IAC with MRQ	100	1.768	75	176.768
IAC with IMQ	100	1.004	75	176.004

2) Network Occupancy: We define the network occupancy in information brokerage systems as total traffic demand over total link capacity. We calculate the network occupancy of a link l as $latency(l) \times TotalTraffic$. As we described, in the backward answer forwarding phase, there is no difference among different architectures, as long as the data, access control policies and queries remain the same. Therefore, we focus on query forwarding phase.

First of all, we compute the average size of queries through 500 queries and obtain the average length as 30 Bytes per query. Next, we assume all queries are enclosed in TCP packets, which brings an header of 40 Bytes. Then, we calculate the network occupancy as:

$$\begin{aligned} NetworkOccupancy &= latency(l) \times TotalTraffic \\ &= latency(l) \times QueryLength \times QueryFrequency \\ &= 100ms \times (30+40)Bytes \times QueryFrequency \end{aligned}$$

As shown in Figure 12, query forwarding consists two network transition phases: (1) from user to broker and (2) from broker to data source. Since 25% of the queries are denied by access control policy and dropped at brokers, the overall network occupancy of IAC forwarding phase is: $100 \times 70 \times (QueryFrequency + QueryFrequency \times 75\%)$. While the overall network occupancy of SAC and EAC forwarding phase is: $100 \times 70 \times QueryFrequency \times 2$. Comparing

[2]http://www.internettrafficreport.com/samerica.htm#graphs.

the two results above, we can see that the IAC architecture saves 12.5% of overall network occupancy.

B. System-wide Security

In information brokerage systems, security is not only a database concern as in the traditional DBMS system, but also a system concern. The overall security of information brokerage systems is not limited to prohibiting users from accessing unauthorized data, rather, it provides a broader concept as the security of the whole system, where DBMS lies at the boundary. The system-wide security benefits from the early denial of suspicious actions and intrinsic replication among brokers.

1) Prohibition of Unauthorized Users: As whole system, suspicious actions should be detected and denied as early as possible; i.e., at the entrance of the system, instead of letting it walk around the core system (brokerage network) and reach the far boundary (designated data server) to be rejected there. However, in traditional information brokerage networks, brokers do not carry any access control function. By sending fake queries to the system, any user (unauthorized or even unregistered user) could bring in risk. For instance, let us assume data source DS_A holds sensitive information (e.g., `//creditcard` nodes) and data source DS_B holds public data (e.g., $//person$ nodes, but not `//creditcard` node). In a traditional brokerage system, a low-level user or an attacker could send a "snooping query" (say $//creditcard$) to trace and locate DS_A, where the query reaches and gets rejected. In this way, one can get a whole picture of the system such as where the servers are and what data they have by keep sending these snooping queries, and do further after successfully finding out the locations of sensitive information.

On the contrary, our in-network access control approach conceals servers that may carry sensitive data (such as DS_A) and blocks potential misfeasance at the brokers. Unauthorized users are isolated from entering the system, therefore significantly reduce their possibility of conducting harmful activities. Thus, it brings more overall system security.

2) Resistance against DoS Attacks: Moreover, our in-broker brokerage system provides a full replication of access control and location information among all the brokers, which brings higher robustness to the whole system. In traditional information brokerage systems, attackers could block a portion of data sources by DoS attacks. Since the security check is at the DBMS end, the attackers could exhaust the network access and the system resource of the target data server by sending a huge number of identical (or similar) queries which have no access right to the requested data. In our in-broker access control approach, not only the DoS attacking data cannot reach the data server but also the broker can easily recovery with the help of other brokers. However, compared with databases (relational tables or XML trees), the size of access control rules is minimum. In our in-network access control system, it is practically applicable to maintain a full version of access control rules at each broker, i.e. access control function components are fully replicated at each broker. In this way, attackers are not able to block-out a portion of data, since their fake queries are mostly closed-out at the brokers. The brokers endure the incoming attacks, while the brokerage network and data sources are successfully protected. To turn down the system, attackers need to successfully DoS all the brokers. This is practically impossible considering the number of brokers in the system. Since the broker only holds the access control and location information, replication at the broker level is not as expensive as the data level replication in other two architectures. However, the concern of the replication cost is one reason of the multi-hop brokering exploration in our future work.

3) Trust of brokers.: Another concern of pushing access control to the brokers is the trust level of the brokers. It is reasonable to assume the brokers have a certain level of trust in intra-organizations brokerage systems, and are only partially trusted in inter-organizations brokerage systems. For the latter circumstance, we should notice that the brokers could be hacked (by outsiders) or abused (by insiders) even without access control enforcement mechanism. In respond to this, we can use dual access control (i.e., double-check or validate if an access control policy is correctly enforced at the data source side) and ene-to-end auditing systems to help monitoring brokers' behavior. Since the in-broker access control is a bonus of the query forwarding process considering the performance which we will discuss later and only the passed queries experience the second security check at the data source side, the dual access control does not greatly hurt the overall query response performance and is an acceptable solution.

VIII. Conclusion

In this paper, we focus on access control issues in XML information brokerage systems, where end-users send in queries without knowing where data is actually stored, and brokers take the responsibility to locate the data sources and forward the queries. We propose a general framework that categories access control approaches into three architectures, namely SAC, EAC and IAC. We show that IAC architecture is desired in terms of network efficiency and robustness. However, due to limitations of access control enforcement mechanisms, none of existing takes IAC architecture. In this paper, we adopt an access control mechanism named QFilter, which was previously proposed by us. By constructing a QFilter for each role and sit it in the brokers, we developed the first In-Network Access Control approach, which pulls access control out of data sources towards the users to enjoy all the benefits of IAC architecture. Observing the great extent of similarities existing between access control policies of different roles, we further optimize the first approach by merging QFilters of different roles into one. Moreover, we propose and NFA based Indexer for brokers to efficiently locate data sources for user queries. We finally merge NFA based Indexer into Multi-Role QFilter to obtain Indexed Multi-Role QFilter. Through detailed experiments, we demonstrate and compare the performance of all structures and approaches.

REFERENCES

[1] Y. Papakonstantinou and V. Vassalos, "Architecture and implementation of an XQuery-based information integration platform," in *IEEE Data Eng. Bull.*, vol. 25, 2002.

[2] F. Li, B. Luo, P. Liu, D. Lee, P. Mitra, W.-C. Lee, and C.-H. Chu, "In-broker access control: Towards efficient end-to-end performance of information brokerage systems," in *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (SUTC'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 252–259.

[3] T. W. Yan and H. Garcia-Molina, "The SIFT information dissemination system," *ACM TODS*, vol. 24, no. 4, pp. 529–565, 1999.

[4] Y. Diao, S. Rizvi, and M. J. Franklin, "Towards an Internet-scale XML issemination service," in *VLDB*, Toronto, 2004.

[5] A. C. Snoeren, K. Conley, and D. K. Gifford, "Mesh-based content routing using XML," in *Symposium on Operating Systems Principles*, 2001, pp. 160–173.

[6] R. Chand and P. A. Felber, "A scalable protocol for content-based routing in overlay networks," in *IEEE International Symposium on Network Computing and Applications*, Washington D.C., 2003, p. 123.

[7] C. Y. Chan, W. Fan, P. Felber, M. N. Garofalakis, and R. Rastogi, "Tree pattern aggregation for scalable xml data dissemination." in *VLDB*, 2002.

[8] G. Koloniari and E. Pitoura, "Content-based routing of path queries in peer-to-peer systems." in *EDBT*, 2004.

[9] "Websphere application server network deployment," http://www-306.ibm.com/software/webservers/appserv/was/ network/edge.html.

[10] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong, "Access control in collaborative systems," *ACM Comput. Surv.*, vol. 37, no. 1, 2005.

[11] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati, "A fine-grained access control system for XML documents," *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 2, pp. 169–202, 2002.

[12] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[13] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati, "Design and implementation of an access control processor for XML documents." *Computer Networks*, vol. 33, no. 1-6, pp. 59–75, 2000.

[14] E. Bertino and E. Ferrari, "Secure and selective dissemination of XML documents." *ACM TISSC*, vol. 5, no. 3, pp. 290–331, 2002.

[15] S. Godik and T. Moses, "eXtensible Access Control Markup Language 1.0," OASIS Specification Set, Feb 2003.

[16] M. Kudo and S. Hada, "XML document security based on provisional authorization," in *CCS*. New York, NY, USA: ACM Press, 2000, pp. 87–96.

[17] T. Yu, D. Srivastava, L. V. S. Lakshmanan, and H. V. Jagadish, "Compressed accessibility map: Efficient access control for XML," in *VLDB*, China, 2002, pp. 478–489.

[18] S. Cho, S. Amer-Yahia, L. V. S. Lakshmanan, and D. Srivastava, "Optimizing the secure evaluation of twig queries." in *VLDB*, China, 2002, pp. 490–501.

[19] B. Luo, D. Lee, W.-C. Lee, and P. Liu, "QFilter: Fine-grained run-time XML access control via NFA-based query rewriting," in *ACM CIKM*, Washington D.C., USA, nov 2004.

[20] G. Miklau and D. Suciu, "Containment and equivalence for an XPath fragment," in *PODS*, Wisconsin, 2002, pp. 65–76.

[21] M. Murata, A. Tozawa, and M. Kudo, "XML access control using static analysis," in *ACM CCS*, Washington D.C., 2003.

[22] W. Fan, C.-Y. Chan, and M. Garofalakis, "Secure xml querying with security views," in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 2004, pp. 587–598.

[23] S. Mohan, J. Klinginsmith, A. Sengupta, and Y. Wu, "Acxess - access control for xml with enhanced security specifications," in *ICDE '06*. Washington, DC, USA: IEEE Computer Society, 2006, p. 171.

[24] S. Mohan, A. Sengupta, and Y. Wu, "Access control for xml: a dynamic query rewriting approach," in *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*. New York, NY, USA: ACM Press, 2005, pp. 251–252.

[25] B. Luo, D. Lee, W.-C. Lee, and P. Liu, "Deep set operators for XQuery," in *SIGMOD Workshop on XQuery Implementation, Experience and Perspectives*, Baltimore, USA., 2005.

[26] A. Schmidt, F. Waas, S. Manegold, and M. Kersten, ""The XML Benchmark Project"," INS-R0103, CWI, Tech. Rep., April 2001.

[27] B. Choi, "What are real dtds like?" in *WebDB*, 2002.

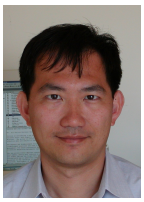
Fengjun Li is a Ph.D. candidate in the College of Information Sciences and Technology at Penn State University. She received her MPhil. degree in Information Engineering from the Chinese University of Hong Kong in 2004, and a B.S. degree from the University of Science and Technology of China in 2001. Her research interests include distributed system security and privacy, database security, and network security.

Bo Luo is a Ph.D. candidate at the College of Information Sciences and Technology, Penn State University. He received his M.Phil. degree in Information Engineering from the Chinese University of Hong Kong in 2003, and a B.S. degree from the University of Science and Technology of China in 2001. His current research focuses on XML and relational database systems, especially security and privacy issues.

Peng Liu is an associate professor of College of Information Sciences and Technology and director of the Cyber Security Lab at Penn State University. He received his Ph.D. degree in Information Technology from George Mason University in 1999, the M.S. and B.S. degrees from the University of Science and Technology of China in 1996 and 1993. His research interests include survivable systems, network security, systems security, privacy, e-commerce, eHealth and cyber infrastructures.

Dongwon Lee is an assistant professor with the College of Information Sciences and Technology at Penn State University. He holds a Ph.D. degree in Computer Science from University of California at Los Angeles in 2002, a M.S. from Columbia University in 1995, and a B.S. from Korea University in 1993. He is interested in conventional database research agenda, XML data model and schema/query languages, information retrieval, digital library, citation analysis and bibliometrics.

Prasenjit Mitra is an assistant professor in the College of Information Sciences and Technology at Penn State University. He received his Ph.D. degree in Electronic Engineering at Stanford University in 2004, a M.S. in Computer Science at the University of Texas at Austin in 1994, and a Bachelor degree from the Indian Institute of Technology in 1993. His primary research focus is on issues related to information integration and interoperation.



Wang-Chien Lee is an associate professor in the Department of Computer Science and Engineering, Penn State University. He received a Ph.D. degree from Ohio State University in 1996, a M.S. degree from Indiana University in 1989, and a B.S. degree from National Chiao-Tung University (Taiwan) in 1985. His research interests include mobile and pervasive data access, location-based service, peer-to-peer computing, wireless sensor networks, mobile ah hoc networks, XML access control and object-oriented database.

Chao-Hsien Chu is a professor in the College of Information Sciences and Technology at Penn State University. He received the Ph.D. degree in Business Administration from the Penn State University in 1984, an MBA from Tatung Institute of Technology, and a B.E. in Industrial Engineering from Chung Yuan University. His research interests include information and cyber security especially in wireless security, intrusion detection, security & risk management and cyber forensics.