In-broker Access Control: Towards Efficient End-to-End Performance of Information Brokerage Systems

Fengjun Li Bo Luo Peng Liu Dongwon Lee Prasenjit Mitra Wang-Chien Lee Chao-Hsien Chu
The Pennsylvania State University, University Park, PA 16802, USA
{fengjun, bluo, pxl20, dongwon, pum10, wul2, chc4}@psu.edu

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. 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 argue that query brokering and access control are **not** *two orthogonal issues because access control deployment strategies can have a 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.*

1. 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 multinational 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.

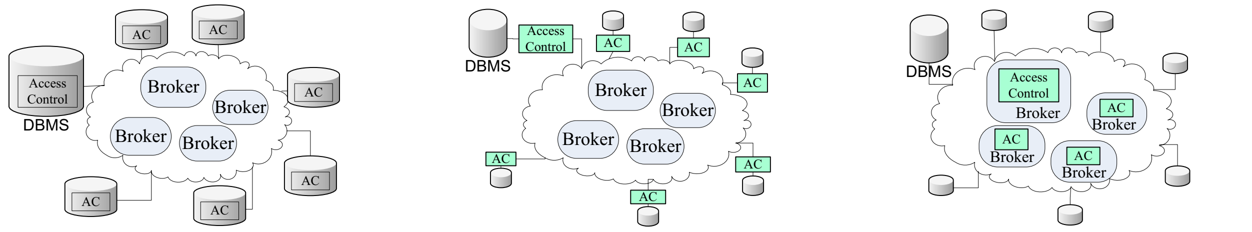
Although the Internet and various virtual private networks provide good data communication links, there are major challenges in (a) achieving scalable, agile and secure remote access of distributed data; (b) handling the heterogeneity among data management systems and data formats which are not always 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. For example, classic distributed database cannot meet the challenges, since they require a static "global" database schema that is fully structured.

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 [17] 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

Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'06)
0-7695-2553-9/06 $20.00 © 2006 IEEE



(a) Embedded Access Controls (EAC) (b) Source-side Access Controls (SAC) (c) In-broker Access Controls (IAC)

Figure 1. Three architectures of information brokerage systems.

can significantly improve the performance of memory consumption, end-to-end query directing time and network occupancy without hurting the system-wide security.

2. Information Brokerage System Architecture

Consider an *information brokerage system* where sensitive information is shared among geographically distributed participants (e.g., users and data sources). 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.

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 1 (a) illustrates this architecture, named as *Embedded Access Controls (EAC)*.

On the contrary, some recent proposals attempt to pull access controls out of DBMS. For instance, in [13], we show that access controls can be supported via query rewriting outside of DBMS, thereby de-coupling 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 1(b) illustrates this architecture, named as *Source-side Access Controls (SAC)*.

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 1(c) illustrates this architecture, namely *In-broker Access Controls (IAC)*. 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. We will discuss this in more detail in Section 6.

3. Background

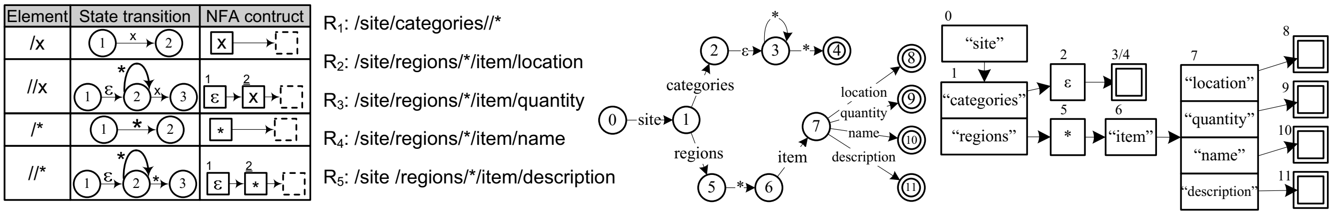
3.1. XML Access Control Model

In this paper, we adopt the fine-grained XML access control model similar to [8], and incorporate Role Based Access Control [18]. 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). Nodes are inaccessible by default, and negative rules take precedence upon positive rules in conflicts.

3.2. Introduction to QFilter

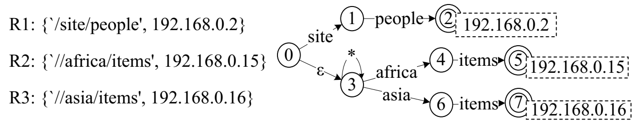
One of the recent developments of XML access control is to enforce access controls on input queries [16, 13]. In this section, we introduce a state-of-the-art technique, called QFilter [13], that we recently proposed. QFilter captures a set of access control rules using a Non-deterministic Finite Automata (NFA), and re-writes parts of incoming query Q that violate the access rights, to yield a safe query Q'.

QFilter Construction. Four basic building blocks (`/x`, `/*`, `//x`, `//*`) of XPath expression are used to construct NFA states, as illustrated in Figure2 (a). The NFA for a complete set of ACR (i.e., XPath expressions in ACR) is formed by linking the states in sequence. QFilter construction process is very similar to regular NFA construction. Let us use ACR of Figure 2 (b) 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: for XPath step `/site`, we create state 0 and a transition on token `site` to state 1; then a transition on token `categories` is created; and so on. Finally, the constructed QFilter is shown in Figure 2 (c).



(a): QFilter building blocks (b): Sample ACR (*object* only) (c): QFilter state transition map (d): QFilter for rules in (b)

Figure 2. An example of QFilter


R1: {`/site/people', 192.168.0.2}
R2: {`//africa/items', 192.168.0.15}
R3: {`//asia/items', 192.168.0.16}
0
site
1
people
2
192.168.0.2
ε
*
3
africa
4
items
5
192.168.0.15
asia
6
items
7
192.168.0.16

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

Figure 3. An example of NFA-based Indexer

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 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 [1].

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 $/site/regions/*/item/location$".

4. Approaches for In-broker Access Control

4.1. 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 3(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 3, 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 3 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.

4.2. The Multi-Role QFilter Approach

4.2.1 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.

4.2.2 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

[1] To be more strict, "DEEP UNION" should be used [14]

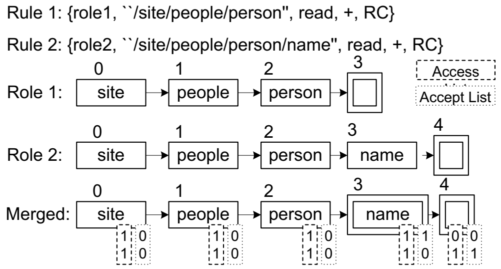
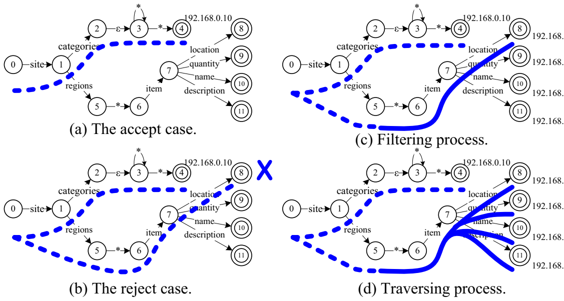


Figure 4. MRQ Construction.



(a) The accept case.

(b) The reject case.

(c) Filtering process.

(d) Traversing process.

Figure 5. IMQ Construction.

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 4 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 4) 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.

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.

4.3. Indexed Multi-Role QFilter Approach

4.3.1 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.

IMQ Construction. As described in Section 4.1, 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 5 (we assume that MRQ is already constructed): (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; (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.

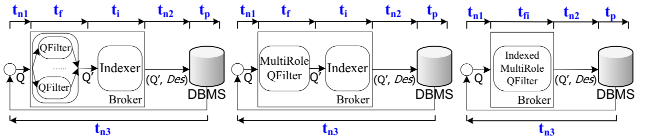
5. Experimental Validation

We have implemented the three brokering approaches (shown in Figure 6) proposed in Section 4. 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.

5.1. Query filtering

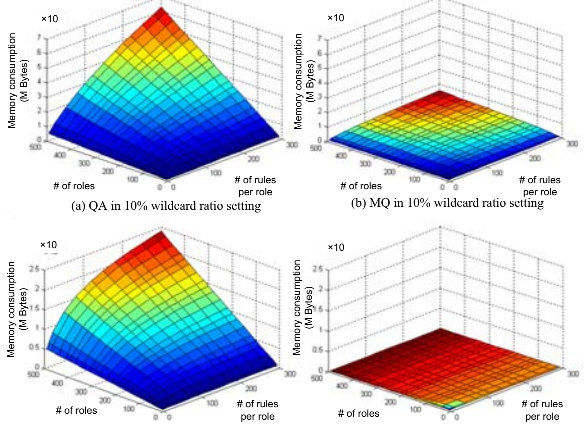
Settings. We use the well-known XML benchmark XMark [19] DTD. It defines 77 elements and 16 attributes for an on-line auction scenario. In rule XPath generation,

$t_{n1,n2,n3}$: network latency; t_i: query indexing; t_f: query filtering; t_p: query evaluation.



(a) QFilter Array + Indexer (b) Multi-Role QFilter + Indexer (c) Indexed Multi-Role QFilter

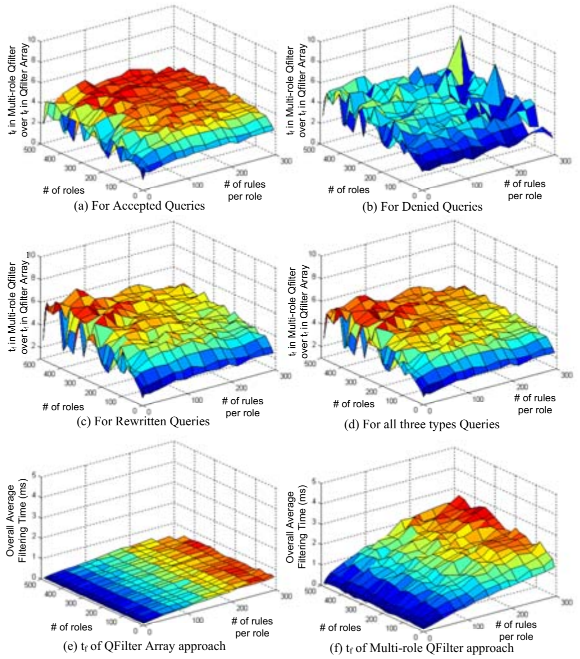
Figure 6. Three brokering approaches.



(a) QA in 10% wildcard ratio setting (b) MQ in 10% wildcard ratio setting

(c) QA in no wildcard ratio setting (d) MQ in no wildcard ratio setting

Figure 7. Memory usage of QA and MRQ.



(a) For Accepted Queries (b) For Denied Queries

(c) For Rewritten Queries (d) For all three types Queries

(e) t_f of QFilter Array approach (f) t_f of Multi-role QFilter approach

Figure 8. Query filtering time of QA and MRQ.

maximal depth of the XPath expressions is set to 6 [6]. 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 7: rules in Figure 7(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 sharing. In Figure 7(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 7 (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. In Figure 8 (a) to (c), we show the ratio of query processing time in MRQ approach over query processing time in QA, for accepted, denied and rewritten queries, respectively. Figure 8 (d) shows this ratio of the average of all queries. Figure 8(e) and (f) 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.

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.

5.2. Query filtering and indexing

In this experiment, we compare the three in-broker access control approaches, as shown in Figure 6: 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)

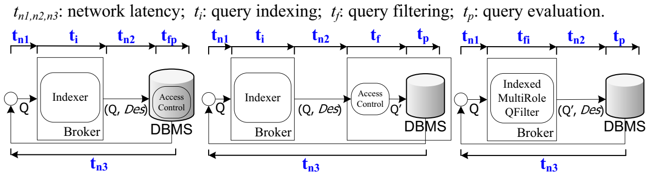
Table 1. Compare the memory and query brokering time of three in-broker approaches.

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

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 1. 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 1. 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.



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

Figure 9. End-to-end query answering time

6. Architecture Level Analysis

In IAC architecture, if access control rules allow a query to access all/partial of the requested XML content (accepted/rewritten query), the original or rewritten query will be forwarded to the query indexer. Otherwise (rejected query), it will be dropped at the broker and/or a error message is returned to the user. In this way, users get denial response faster for rejected queries as well as a deduction in overall response time. At the same time, less network resources is consumed by only directing accepted and rewritten queries to the data sources.

6.1. End-to-End Performance Improvement

Murata et al have conducted experiments to investigate the accepted and denied queries [16]. 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 experiment accordingly.

End-to-End Query Directing Time. We define the end-to-end query directing time as the summation of query filtering time (t_f), indexing time (t_i), and network latency (t_n), as shown in Figure 9. Since same user queries, access control rules and DBMS are used across all architectures, the processing time t_p and backward network latency t_{n3} should the same. 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. Compared with the results in experiment 2 (Table 2), it is clear that the network latency dominants, and thus the performance under IAC architecture is much better than the one using SAC architecture.

Network Occupancy. Defined as total traffic demand over total link capacity, we calculate the network occupancy of a link l as $latency_l \times total\_traffic(inByte)$. We measure the size of 500 queries in S_Q and take the average 30 Bytes as the value for one query. Further assume all queries are enclosed in TCP packets, which brings an additional header of 40 Bytes. Then, we calculate the network occupancy as $latency(100ms) \times traffic(30 + 40Bytes) \times$

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

Table 2. End-to-end query answering time.

Approach/Time(ms)	t_{n1}	t_f+t_i	t_{n2}	overall
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

$No.\_of\_queries$. Since 25% queries are denied, the saving of IAC over SAC and EAC in network occupancy is: $(100 \times 70 \times 500 + 100 \times 70 \times 500 \times 75\%)/(100 \times 70 \times 500 \times 2) = 87.5\%$.

6.2. 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.

As whole system, suspicious actions should be detected and denied 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 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). In a traditional brokerage system, a low-level user (e.g. the 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. In the contrary, our in-network access control approach conceals servers with sensitive data (such as DS_A) and blocks potential misfeasance at the brokers. Thus, it brings more overall system security.

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.

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.

7. Related Work

Publish/Subscribe systems (e.g., [22, 9]) are based on events and provide many-to-many communication between event publishers and subscribers. What we have proposed in not a publish/subscribe system for its spontaneous query answering capability. As an XML-based overlay network, [20] proposed a mesh-based overlay network that supports XML queries. In [4] XML content-based routing is addressed using the query aggregation scheme given in [3]. In [11], 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 [4, 1]. 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. [21] 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.

In the proposed XML brokerage system, we used the access control model proposed by [8, 18]. However, since ours is not tightly coupled with one specific model, our proposed techniques can be applied to other access control models (e.g., [7, 2, 10, 12]). As to enforcing XML access controls, by and large, existing approaches either use "views" (e.g., compressed accessibility map of [23]) or rely on the underlying XML engine (e.g., [5]). Our proposal is based on the QFilter – 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 [15], 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.

8. 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 captures existing and proposed access control approaches into three categories, 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] Websphere application server network deployment. http://www-306.ibm.com/software/webservers/appserv/was/network/edge.html.

[2] E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM TISSC*, 5(3):290–331, 2002.

[3] C. Y. Chan, W. Fan, P. Felber, M. N. Garofalakis, and R. Rastogi. Tree pattern aggregation for scalable xml data dissemination. In *VLDB*, pages 826–837, 2002.

[4] 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*, page 123, Washington D.C., 2003.

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

[6] B. Choi. What are real dtds like? In *WebDB*, 2002.

[7] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access control processor for XML documents. *Computer Networks*, 33(1-6):59–75, 2000.

[8] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202, 2002.

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

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

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

[12] M. Kudo and S. Hada. XML document security based on provisional authorization. In *CCS*, pages 87–96, New York, NY, USA, 2000. ACM Press.

[13] 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.

[14] 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.

[15] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *PODS*, pages 65–76, Wisconsin, 2002.

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

[17] Y. Papakonstantinou and V. Vassalos. Architecture and implementation of an XQuery-based information integration platform. In *IEEE Data Eng. Bull.*, volume 25, 2002.

[18] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

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

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

[21] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37(1), 2005.

[22] T. W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM TODS*, 24(4):529–565, 1999.

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