In-broker access control: A new access control deployment strategy towards optimal end-to-end performance of information brokerage systems

Fengjun Li Bo Luo Dongwon Lee Prasenjit Mitra
Wang-Chien Lee Chao-Hsien Chu Peng Liu
The Pennsylvania State University, University Park, PA 16802, USA

Abstract

An XML brokerage system is a distributed XML database system that comprises of 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 (in general) neglected by existing information brokerage systems; and 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 of an information brokerage system into the "heart" of the system. We design and evaluate the fist 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.*

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 enterprises that are becoming ever more globalized and distributed. Multiple divisions cooperate inside large multinational enterprises as day's routine. For example in GM, to maintain a proper stock level of the parts, people in the supply management division need to check the sale information (of a set of car models) gathered and managed by people in the world-wide sales divisions. In such information sharing systems, the data gathered by a specific division unit 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 in addition to this division.

Although the Internet and various virtual private networks provide good data communication links, there are major challenges (a) in achieving scalable, agile, precise and secure remote access of locally managed data; (b) in handling the heterogeneity among the local data management systems and the data formats (used in these systems) which are not always structured and may be incompatible with each other; (c) in handling the dynamics of modern business applications (where new schema elements may emerge everyday); and (d) in location discovery. For example, classic distributed database systems cannot meet the challenges, since they require a fairly static "global" database schema and they require all data objects to be fully structured.

To tackle these challenges, mediation and federation based information brokering technologies have been extensively studied. Recently, eXtensible Markup Language (XML) has been shown to be a promising solution [20], by integrating data in incompatible formats using a semantic-preserved form. An XML-based information brokerage system is a distributed XML database system that comprises of data sources and brokers which, respectively, hold (conceptually) 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 and particularly the 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 orthog-

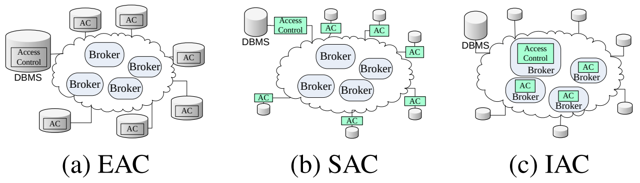


Figure 1. Three architectures of information brokerage system.

onal 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 (in general) neglected by existing information brokerage systems; and data source side access control deployment is taken-for-granted as the "right" thing to do. In this paper, we challege 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. In particular, we propose the first in-broker access control deployment strategy where access control is "pushed" from the boundary of an information brokerage system into the "heart" of the system. We design and evaluate the fist in-broker access control scheme for information brokerage systems, and the experimental results show that in-broker access control may significantly improve the performance of memory consumption, query response time and network occupancy as well as system-wide security. Our experimental results indicate that information brokerage system builders should treat access control as a system issue as well.

2 Architectures of Information Brokerage Systems

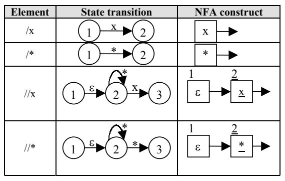
Consider an *information brokerage system* where sensitive information is being shared among geographically distributed participants (e.g., users, data sources and brokers). To make the exposition simple, let us assume that each broker has a full knowledge of whereabouts of stored data. Therefore, each broker may direct an inquiry to relevant data sources without consulting other brokers (i.e., single-hop brokering). Since the 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 system, the job of security enforcement is solely left upon the shoulder of DBMS. For instance, administrators store access controls inside DBMS, and any query first needs to pass access controls by DBMS before it is processed by engines. 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 of the recent proposals attempt to pull the access controls out of DBMS. For instance, in [15], we show that access controls can be supported via query rewriting method 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 supported even if data are stored in a DBMS without such a capability. For instance, none of the commercially available XML databases are capable of supporting access controls. However, using our proposed QFilter, access controls can be supported outside of DBMS. Figure 1(b) illustrates this architecture, named as *Source-side Access Controls (SAC)*. Intrigued by this pulling out of DBMS procedure, we further push the access controls to the brokers, from the boundary of brokerage systems into the "heart" of the system. In this way, security check is enforced when user accesses the brokerage network. Figure 1(c) illustrates this architecture, named as *In-broker Access Controls (IAC)*. We realized that both query brokering and access controls are *not* two orthogonal issues. By integrating the two issues properly, the whole system benefits from this integrated design and improve the end-to-end performance.

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. If the rules allow the user to access all the requested XML content, which means the original query is accepted by the rules, it will be send to the query indexer. If the rules allow the user to access a portion of the requested content, the original query will be rewritten into a safe one, which yields accessible content only. The rewritten query will be send to the query indexer. If the rules does not allow the user to access any of the requested content, which means the query is fully rejected, it will be dropped at the broker and/or a error message is returned to the user. Then 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 the accepted and rewritten queries to the data sources. The improvement in end-to-end performance will be discussed in more detail in Section 5.

3 Background

XML Access Control Model. In this paper, we adopt the fine-grained XML access control model similar to [10],



(a)

R_1: /site/categories//*

R_2: /site/regions/*/item/location

R_3: /site/regions/*/item/quantity

R_4: /site/regions/*/item/name

R_5: /site /regions/*/item/description

(b)

Figure 2. (a) Qfilter building blocks; (b) Sample ACRs (*object* part only).

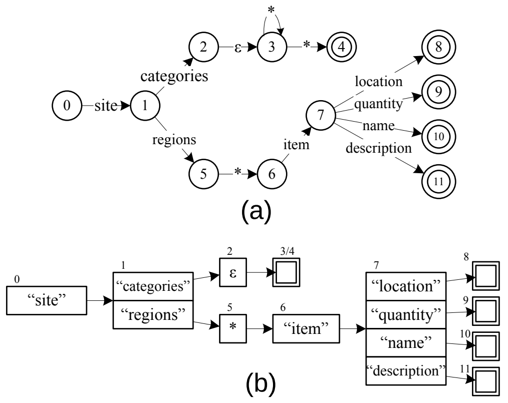


Figure 3. QFilter example: (a) State transition map and (b) NFA transition.

and incorporate Role Based Access Control [21] to make it more pragmatic. In our model, database 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): *access control rule* = {*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 either access "granted" or "denied," respectively; and (5) *type* $\in \{LC, RC\}$ refers to either "Local Check" (i.e., authorization is applied to only attributes or textual data of context nodes), or "Recursive Check" (i.e., authorization is applied to current nodes and propagated to all the descendants), respectively. Nodes are inaccessible by default, and negative rules take precedence upon positive rules in conflicts.

QFilter: Enforcing XML Access Controls via Query Re-writing. One of the recent developments in XML access controls is to enforce XML access controls by "re-writing" input queries (e.g., [19, 15]). In this section, we introduce a state-of-the-art technique, called QFilter [15], that we recently proposed. The QFilter captures a set of access control rules (ACR) using a Non-deterministic Finite Automata (NFA), and re-writes any parts of incoming query Q that violate the given access rights, to yield a safe query Q'.

QFilter Construction. Four basic building blocks (`/x`, `/*`, `//x`, `//*`) of XPath expression are used to construct NFA fragments, 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: we create state 0 and a transition on token `site` to state 1 for element `/site`; then a transition on token `categories` is created on `/categories`; and so on. Finally, the constructed QFilter is shown in Figure 3.

QFilter Execution. In the context of access controls, QFilter captures ACR^+ and ACR^-. For an input XPath query Q, QFilter has three types of operations during the execution: (1) **Accept**: If answers of Q are contained by that of ACR^+ (i.e., Q asks for answers granted by ACR^+) and disjoint from that of ACR^- (i.e., Q does not ask for answers blocked by ACR^-), then QFilter accepts the query as it is: $Q' = Q$; (2) **Deny**: If answers of Q are disjoint from that of ACR^+ (i.e., no answers to Q are granted by ACR^+) or contained by that of ACR^- (i.e., all answers to Q are blocked by ACR^-), then QFilter rejects the query outright: $Q' = \emptyset$; (3) **Rewrite**: if only a partial answer is granted by ACR^+ or partial answer is blocked by ACR^- , QFilter rewrites Q into the ACR-obeying output query Q'. At the end, the following property is guaranteed: Q' is (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} is the ACR^+ in Figure 3. That is, a role *user* can read only the `name` and `location` elements under `item`. Then, for the input query Q : $//item[1]/*$, the QFilter of Figure 3 would yield the following re-written query Q' : $/site/regions/*/item[1]/name$ UNION $/site/regions/*/item[1]/location$.

4 Approaches for In-broker Access Control

4.1 Brokering Indexer

In an XML data brokerage system, users send queries without knowing where the data is stored. Brokers have

[1]To be more strict, "DEEP UNION" should be used [16]

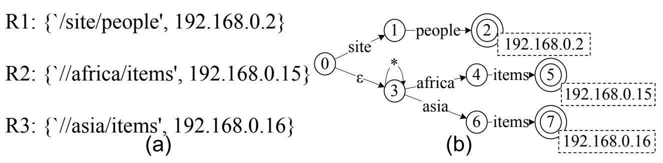

R1: {'/site/people', 192.168.0.2}
R2: {'//africa/items', 192.168.0.15}
R3: {'//asia/items', 192.168.0.16}
(a)
site
people
*
ε
africa
items
asia
items
192.168.0.2
192.168.0.15
192.168.0.16
(b)

Figure 4. An NFA-based indexer in (b) is constructed with index rules in (a).

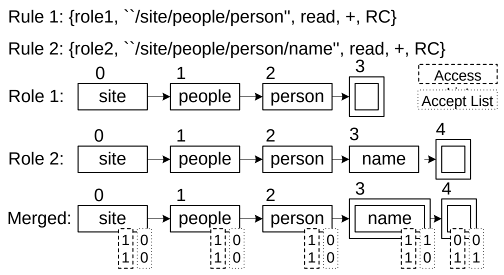

Rule 1: {role1, "/site/people/person", read, +, RC}
Rule 2: {role2, "/site/people/person/name", read, +, RC}
Role 1:
site
people
person
Access
Accept List
Role 2:
site
people
person
name
Merged:
site
people
person
name

Figure 5. QFilter Array of two roles are merged into one Multi-role QFilter.

the information about the physical distribution of the XML documents. In our setting, a query is routed using single-hop brokering. Thus, given an XML query, the broker is able to determine the location of requested content. Note that multi-hop brokering might be employed in lower layers, e.g., if the destination is identified by IP addresses, IP layer query forwarding is multi-hop.

The index information is described as R^{index}={object, destination(s)} where "destination" is a network address (e.g. IP address) and "object" is an XPath expression. An example is shown in Figure 4(a).

Since the index table look-up process is essentially one-to-many XPath matching, we design a QFilter-like extended NFA to handle it. Similar to QFilter, our system constructs an Query Indexer using the XPath expressions of the index rules. At each accept state, the destination address is attached. For instance, Figure 4(b) shows the state transition map for the index rules in Figure 4(a). The execution (destination lookup) is a token matching and state transiting process, which is very similar to any NFA or QFilter execution. The goal is to match incoming queries with routing rules captured in the NFA. During the execution process, the destination addresses are attached to the query when appropriate. Finally accepted queries are forwarded to the list of destination addresses attached to it. It is possible that a query does not match any routing rule, which means no known data source has the requested data, thus the query is dropped.

4.2 The Multi-role QFilter Approach

4.2.1 QFilter Array

The QFilter approach described earlier is designed for one single role [15]. In a network setting, access control for multiple roles (each has its own *ACR*) is needed. To address this need, one intuitive extension is to use an array of QFilters (called *QFilter Array*), in which each QFilter is constructed, stored and executed independently. When a query is submitted, the role of user is first 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. Although this looks not a problem, practically when large number of roles exists, it soon grow beyond size of main memory, and dramatically downgrades the system performance. To tackle this problem, we introduce Multi-role QFilter.

4.2.2 Multi-role QFilter

We observe that there is great similarity in access control rule sets for different roles, therefore their QFilters are also similar. The idea of *Multi-role QFilter* (MRQ) is to merging 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*) since its looking up complexity is $O(1)$.

MRQ Construction. We construct MRQ in a similar way as single QFilters, and enhance it by annotating each state with two bitmaps: *access_list* and *accept_list*, with each bit representing a Boolean value. 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 5 shows an example: there are two roles, each is assigned with an access control rule; 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 5) contains an *access_list* and an *accept_list* 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 none of them is an accept state (the *accept_value*s are 0). The state 3 is the accept state for role 1 only and the state 4 is accessible by role 2 only.

MRQ Execution. Similar to a single QFilter, for an input query $(Q, role\_id)$, MRQ has three types of execution results: **Accept**, **Deny**, or **Rewrite**. During the execution, at each MRQ state, the access right of the role is first checked based on the $role\_id$. If the *access_value* is 0 at a particular state s for a role R, the execution is stopped for role R at state S and the query is not accepted. 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 the data source towards the center of the network - the brokers. Therefore, brokers hold both indexing and access control mechanisms. When incoming query Q is submitted, Multi-role QFilter processes it to safe query Q', then Brokering Indexer locates the data source. Since two mechanisms with similar structure reside at the same place, it is natural to merge them for to improve both storage and processing speed. Therefore, we merge them into one NFA-based structure, namely `Indexed Multi-role QFilter` (*IMQ*), which captures both indexing and access control rules. A query Q sent to IMQ yields the output of $\{Q', \{destination(s)\}\}$, where Q' is the safe query.

IMQ Construction. As described in Section 4.1, the index rule is formatted as R^{index}={object, destination(s)}. The merging process is compromised of three steps: (1)**Construct**: construct a Multi-role QFilter (MQ) using ACR; (2) **Filtering**: executing the XPath of $R^{index}.object$ in MR; and (3) **Attach**: attach $R^{index}.destination(s)$ to the current accept state, as well as all the descendent accept states.

Instead of giving the exhausted algorithm, we use an example to show the construction of IMQ. As shown in figure 6 (we assume that MQ is already constructed): (a) rule {/site/categories//*, 192.168.0.10} reaches state 4, indicating all /site/categories//* data are accessible and are located at 192.168.0.10); (b) rule {/site/regions/item/payment, 192.168.0.5} does not reach any accept state, indicating no user is allowed to access the node although it is located at 192.168.0.5); (c) and (d): when rule {/site/regions, 192.169.0.13} is executed, the XPath expression stops at state 5, then we attach its destination to all the descendant accept states, e.g. 8, 9, 10 and 11. This means, although 192.169.0.13 holds entire /site/regions node, only some descendants are defined accessible by ACR.

IMQ Execution. For an input query $(Q, role\_id)$, IMQ execution is almost identical to Multi-role QFilter execution, except that the appropriate *destination* addresses are attached to accept and rewritten queries. Finally the accepted/rewritten queries are forwarded to all the attached destinations.

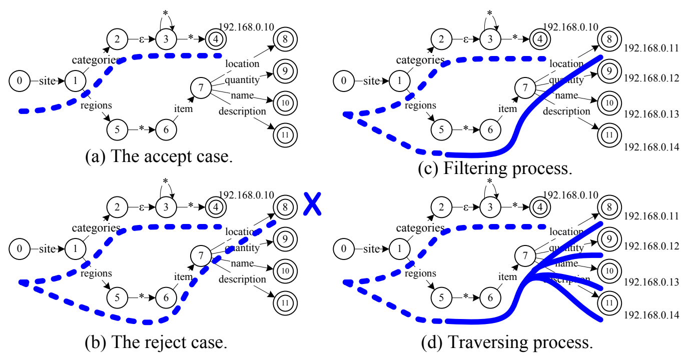


Figure 6. IMQ Construction.

4.3.2 Performance Analysis

The performance of Indexed Multi-role QFilter can be qualitatively analyzed over two parameters: memory and processing time. In terms of memory: (1) Brokering Indexer is removed; (2) each time we attach *destination* to a Multi-role QFilter state, only four bytes (if we use IP address to identify destinations) are added. However, considering both the savings and the additional cost, on the whole we achieve significant storage savings. In terms of query processing time, incoming queries only need to traverse one NFA, instead of two NFAs in series.

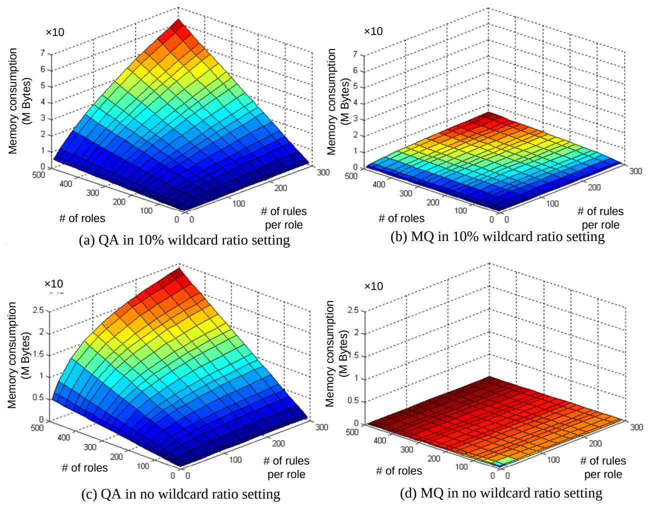
5 Experiment and Analysis

We have implemented the three brokering mechanisms proposed in Section 4 in Java. In this section, we present two experiments based on the implementation. In the first experiment, we investigate how memory cost and query filtering time change with parameters (the number of roles and rules per role) in QFilter Array approach and Multi-role QFilter approach. A reasonable setting is then chosen for the second experiment, and we measure the overall memory consumption and query brokering time in three approaches and show the Indexed Multi-role QFilter approach performs best in both.

5.1 Effectiveness of Role-merging

We wish to show how the memory cost and query filtering time change with parameters in both approaches and how much saving achieved by the Multi-role QFilter approach over the QFilter approach at the same setting. The saving is due to the effectiveness of the states sharing among access control rules in the role-merging process.

Settings. We use a DTD from XMark, a well-known XML benchmark [22]. This DTD defines 77 element and 16 attribute types for an on-line auction scenario. The maximal depth of the XPath expressions is set to 6 as suggested in



(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 consumption of QFilter-Array approach and Multi-role QFilter approach under two settings.

[8]. To mostly simulate the real-world cases, synthetic rules are generated by randomly assigning wildcards (* or //) or predicates to each rule at some given ratios. The complete randomness offsets the impact of the rule pattern. Then we vary the number of roles from 10 to 500 and the average number of rules (with 10% wildcard ratio and no predicate) for each role from 5 to 300 and construct the QFilter Array and Multi-role QFilter accordingly. To evaluate the query filtering time, we generate a synthetic query set S_Q of 500 queries (with 10% wildcard and 1 predicate per rule).

Memory Cost. Memory consumption in QFilter Array approach and Multi-role QFilter approach is shown in Figure 7. In experiments shown in Figure 7(a) and 7(b), a wildcard occurs at each state of access control rules at a given ratio 10%. As expected (see Figure 7(a)), memory consumption for QFilter Array is right proportional to the number of roles and increases below-linear with the number of rules per role. This dues to the percentage of state sharing in both dimensions. Sharing can occur among rules in the same QFilter and the more the rules per role, the more possibility the states share; and no sharing among the rules belong to different roles since they are situated in multiple QFilters. Look at Figure 7(b), we can see a significant saving in memory along both dimensions. This is because all rules are contained in a big QFilter-structure, and states share even when they belong to different roles. To show the curve more clear, a setting with no wildcard in each ACR is used and the experiment result is shown in Figure 7(c) and 7(d). Under this setting, only 105 distinct XPath-based ACR are generated and the percentage of sharing is extremely high. In both settings, the memory requirement for Multi-role QFilter is one order of magnitude smaller than that of the QFilter Array.

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 passed to the access control mechanisms: denied, accepted with rewriting, or accepted without rewriting. Accepted queries take a longer time than the denied ones, while the rewritten queries take the longest time. For S_Q, 198 queries are rejected and 302 queries are accepted. The ratio of filtering time (t_f) in Multi-role QFilter over the one in QFilter Array are calculated and grouped in query types (as shown in Figure 8(a)-(c)). Figure 8(d)gives an overall comparison based the average of all three types. It is clear to see that the Multi-role QFilter approach costs about 4 times in filtering time than the QFilter Array approach no matter which type of query is examined. We can also calculate the overall filtering time based on the proportion of the query types and their average filtering time. Figure 8(e) and 8(f) shows that Multi-role QFilter spends 1.5 to 5 times in filtering than QFilter Array.

Analysis. There is a tradeoff between the memory consumption and filtering time. However, the filtering time is in the scale of milliseconds, much smaller than the network latency (in the scale of hundreds of milliseconds). Thus, the impact is not as significant as memory consumption. We conclude that Multi-role QFilter is a better solution than QFilter Array.

Since the impact of *role number* and *rule number* has been examined in the experiment, and the relative performance of Multi-role QFilter approach and QFilter Array approach is investigated in a great extent, we are able to decide a setting which is applicable in real-world cases for the next experiment.

5.2 Effectiveness of the Indexed Multi-role QFilter Approach

Three brokering mechanisms are proposed in Section 4: first using QFilter Array or Multi-role QFilter for access control and then using Indexer for indexing, or using a all-in-one Indexed Multi-role QFilter. In this test, we wish to evaluate the overall performance of these three mechanisms.

Settings. We fix the number of roles to 80 and the number of rules per role to 50, and randomly generate the synthetic access control rules with 10% wildcard ratio and 1 predicate for each rule. For indexing, we generate synthetic XPath-based index paths at 10% wildcard ratio to offset the impact of path pattern. Since we do not consider predicate parsing in our index scheme, we assume no predicate in the index path. Two sets are built, one with 1000 index paths (S_{P1}) and the other with 4000 index paths (S_{P2}). To measure the query brokering time, the same synthetic query set

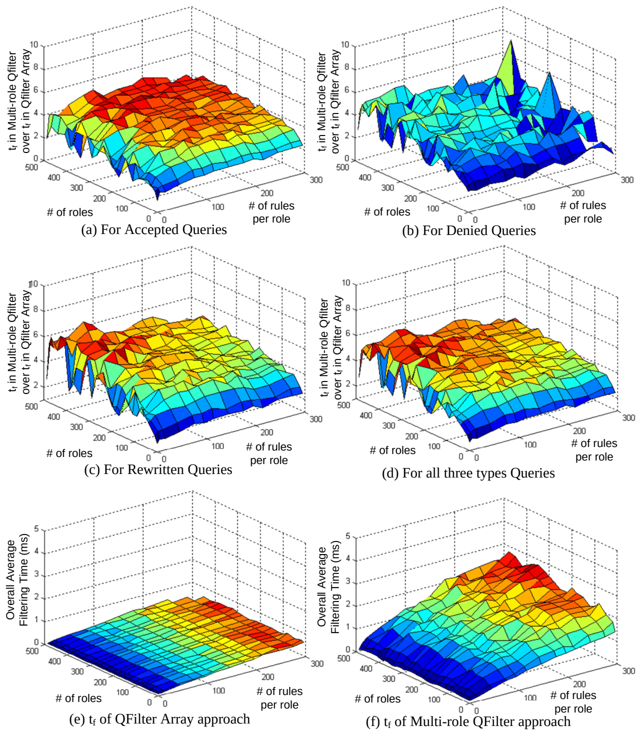


Figure 8. Compare the query filtering time of QFilter Array approach and Multi-role QFilter approach.

S_Q of 500 queries is used.

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 storage respectively. Overall memory consumption of three mechanisms is summarized in Table 1. It can be clearly observed that Indexed Multi-role QFilter requires the least amount of memory, while Multi-role QFilter+Indexer consumes much less than the naive QFilter Array+Indexer. By merging the index paths with the access control rules already in Multi-role QFilter, the Indexed Multi-role QFilter (with 1000 index paths) only requires an additional memory (compare with mechanism MQ+I) of 125KB instead of the original 418KB (used by the Indexer). When the amount of index paths increase to 4000, the saving is much 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 in S_Q 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

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

Approaches (with 1000 indexes)	QA+I	MQ+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**	**MQ+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

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.

6 Architecture Level Analysis

6.1 Analysis on End-to-End Query Directing Time and Network Occupancy

Murata and Kudo have conducted experiments under a dynamic manner to investigate the static accepted queries and static denied queries [19]. 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 is 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), query index time (t_i), and network latency (t_n). Since exactly the same query set reaches the DBMS after the security check, the processing time t_p and backward network latency t_{n3} are the same and are not counted in. General network latency is 200ms [2], so we assume the value of t_{n1} and t_{n2} for a single query is both 100ms. Since 25% queries fail the security check and get rejected at the

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

Table 2. Compare the end-to-end query directing time.

Approach/Time(ms)	t_{n1}	t_f+t_i	t_{n2}	overall
SAC with QA	100	1.014	100	201.014
SAC with MQ	100	1.768	100	201.768
IAC with QA	100	1.014	75	176.014
IAC with MQ	100	1.768	75	176.768
IAC with IMQ	100	1.004	75	176.004

broker in the In-broker Access Control architecture (IAC), the average time of t_{n2} is reduced to 75ms. Based on the results in experiment 2, we calculate the end-to-end query directing time for three brokering mechanisms under two architectures and list the value in Table 2. It is clear that the network latency dominant, 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 $latency(100ms) \times traffic(30 + 40Bytes) \times No.\_of\_queries$. 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 Overall System-wide Security Discussions

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., [26, 2, 11]) 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, [24] 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 [5]. In [13], 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, 1]. Some of recent works has focused on allowing users to specify coherence requirements over data [2, 23]. 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. [25] 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 [10, 21]. However, since ours is not tightly coupled with one specific model, our proposed techniques can be applied to other access control models (e.g., [9, 3, 12, 14]). As to enforcing XML access controls, by and large, existing approaches either use "views" (e.g., compressed accessibility map of [27]) or rely on the underlying XML engine (e.g., [7]). 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 [17, 18], our approach is NFA-based and security-driven. [4], independently developed, bears some similarity to our QFilter approach in that it also uses NFA to process streaming XML data for access controls. 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 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] Websphere application server network deployment. http://www-306.ibm.com/software/webservers/appserv/was/network/edge.html.

[2] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Symposium on Principles of Distributed Computing*, pages 53–61, 1999.

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

[4] L. Bouganim, F. D. Ngoc, and P. Pucheral. Client-based access control management for XML documents. In *VLDB*, pages 84–95, Toronto, Canada, 2004.

[5] 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.

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

[7] 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.

[8] B. Choi. What are real dtds like? In *WebDB*, pages 43–48, 2002.

[9] 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.

[10] 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.

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

[12] S. Godik and T. Moses. eXtensible Access Control Markup Language (XACML) version 1.0. OASIS Specification Set, Feb 2003.

[13] G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In *EDBT*, pages 29–47, 2004.

[14] M. Kudo and S. Hada. XML document security based on provisional authorization. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 87–96, New York, NY, USA, 2000. ACM Press.

[15] B. Luo, D. Lee, W.-C. Lee, and P. Liu. QFilter: Fine-grained run-time XML access control via NFA-based query rewriting. In *13th ACM Int'l Conf. on Information and Knowledge Management (CIKM)*, Washington D.C., USA, nov 2004.

[16] B. Luo, D. Lee, W.-C. Lee, and P. Liu. Deep set operators for XQuery. In *ACM SIGMOD Workshop on XQuery Implementation, Experience and Perspectives (XIME-P)*, Baltimore, MD, USA., 2005.

[17] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 65–76, Wisconsin, 2002.

[18] J. Moffett, M. Sloman, and K. Twidle. Specifying discretionary access control policy for distributed systems. *Comput. Commun.*, 13(9):571–580, 1990.

[19] M. Murata, A. Tozawa, and M. Kudo. XML access control using static analysis. In *ACM conf. on Computer and Communication Security*, Washington D.C., 2003.

[20] Y. Papakonstantinou and V. Vassalos. Architecture and implementation of an XQuery-based information integration platform. In *IEEE Data Eng. Bull.*, volume 25 of *1*, pages 18–26, 2002.

[21] 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.

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

[23] S. Shah, S. Dharmarajan, and K. Ramamritham. An efficient and resilient approach to filtering and disseminating streaming data. In *VLDB*, pages 57–68, 2003.

[24] 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.

[25] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37(1):29–41, 2005.

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

[27] 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.