Efficient and Selective Publishing of Hierarchically Structured Data

by Mohamed Nabeel, Elisa Bertino
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Efficient and Selective Publishing of Hierarchically Structured Data

Mohamed Nabeel, Elisa Bertino
Purdue University,
West Lafayette, Indiana, USA
{nabeel, bertino}@cs.purdue.edu

Abstract

Hierarchical data models (e.g. XML, Oslo) are an ideal data exchange format to facilitate ever increasing data sharing needs among enterprises, organizations as well as general users. However, building efficient and scalable Event Driven Systems (EDS) for selectively disseminating such data remains largely an unsolved problem to date. In general, an EDS has three distinct parties - Content Publishers ($\mathcal{P}$), Content Brokers ($\mathcal{B}$), Subscribers ($\mathcal{U}$) - working in a highly decoupled Publish-Subscribe (PS) model. With a large Subscriber base having different interests and many documents ($\mathcal{D}$), the deficiency in existing such systems lies in the techniques used to distribute (match/filter and forward) content from $\mathcal{P}$ to $\mathcal{U}$ through $\mathcal{B}$. Thus, we propose an efficient and scalable approach to selectively distribute different subtrees of possibly large documents, which have access control restrictions, to different U_i's $\in \mathcal{U}$ by exploiting the hierarchical structure of those documents. A novelty of our approach is that we map subscription routing tables in $\mathcal{B}$ to efficient tree data structures in order to perform matching and other commonly used operations efficiently. $\mathcal{B}$ form a DAG consisting of multiple trees from $\mathcal{P}$ to $\mathcal{U}$. Along with our simple but adequate subscription language, our proposed approach combines policy-driven covering and merging based routing to dramatically reduce the load towards the root of the distribution trees leading to a scalable system. The experimental results clearly reinforce our claims.

Index Terms

Selective Publishing, XML, Publish-Subscribe, Routing

I. Introduction

Hierarchical Data Models (HDMs) naturally capture logical relationships found in enterprise and organizational data and are much more expressive compared to flat data models.The widespread adoption of HDMs such as XML as the de-facto standard to disseminate and/or store content in enterprise Web Services and PS systems is a good indicator of their benefits. Since XML is the most popular HDM, we use XML to illustrate and evaluate our approach. However, the techniques mentioned in the paper can be applied to any HDM. Prior research on XML PS systems has focused on two different problems; the problem of distributing different messages (documents) to different user groups [1], [2], [3], [4], [5], [6], [7], [8], and the problem of disseminating or allowing access to different portions of the same (possibly very large) document to different user groups [9], [10], [11], [12]. The latter, the focus of this paper, is becoming increasingly important in both commercial and collaborative environments. Subscription based content, including news, magazines and multimedia, delivery, stock market surveillance/trade reports, weather data dissemination and business collaborations are some of the applications falling into the domain under our consideration. Ever increasing user base and huge volumes of data are two common denominators of most of these applications.

While approaches in one area of research [9], [10], [11], [12] focus on security of the XML data being disseminated, they fail to achieve the efficiency and scalability required for large user bases and huge amounts of data at the same time. We observe that the bottlenecks in the subscription handling and update notification lead to such inadequacy. A recent research effort [13] has introduced the idea of disseminating XML documents based on their structure at the same time assuring confidentiality and integrity. However, neither efficiency nor scalability of dissemination has been considered. Most existing XML document dissemination approaches [1], [3], [5], [7], [6] use subsets of XPath or XQuery. These approaches have at least a polynomial time complexity [14]. However, it is crucial to realize that certain practical systems such as subscription based content delivery base their access control decisions and subscription granularity not on the actual content itself but on different sections of the content. For example, a highly configurable online news delivery system may allow users to access only certain subsections of the news paper based on the payment they have made and an advanced hospital may allow different employees to access only certain subsections of medical records disseminated based on the role(s) they play. This observation allows one to have a simplified query language at the XML Schema level. In this paper, we exploit both subscription query language and routing techniques to achieve efficient and scalable subscription handling and selective update notification.

In summary, we introduce covering and merging based routing for XML documents along with a novel tree data structure to construct routing tables, in order to perform PS operations efficiently and in a scalable manner. We also introduce a simple, yet expressive enough to fulfill the task at hand, subtree based pattern matching language which allows one to build a much less computationally complex system compared to existing approaches based on XPath or XQuery. Instead of using conventional tree topologies to decide routing paths, we use a better approach to dynamically decide routing paths based on a configurable policy creating more opportunities for covering relationships.

The rest of the paper is organized as follows. Section 2 provides background information on PS systems, and subscription covering and merging. Section 3 describes related work in the areas of Content-Based PS (CBPS) systems, XML data dissemination and related security aspects. Section 4 covers our approach outlining the annotation and encoding scheme, subscription handling and covering/merging based routing. Section 5 describes some of the key algorithms and protocols specifically devised for our approach. Section 6 has experimental results. Section 7 discusses some practical considerations related to real PS systems. Section 8 concludes this paper with possible future work.

II. BACKGROUND

In an EDS based on PS model, each $U_i \in \mathcal{U}$ selectively subscribes to different subtrees of documents $\mathcal{D}$ with some B_i's $\in \mathcal{B}$ possibly based on either a content-based payment scheme and/or access control policies. When P_i's $\in \mathcal{P}$ publish documents to some B_i's, those B_i's, in turn, selectively distribute to other B_i's and finally to U_i's based on the subscriptions. These systems, in general, follow a *push based* dissemination approach, that is, whenever new events/messages arrive, B_i's perform the filtering and distribute the events to requested and legitimate U_i's. PS models provide a highly scalable architecture to distribute events/messages among loosely-coupled entities. Three major types of traditional PS systems have been proposed based on the filtering technique employed [15]: topic-based (events are grouped into channels), content-based (events are matched based on the content of the events/messages) and type-based (events are matched to their programming type). We observe that these traditional techniques solve the problem of sending many documents to many subgroups of U_i's in $\mathcal{U}$. However, those approaches are not directly applicable to the problem of sending many portions (subtrees) of a few large documents to many subgroups.

Figure 1 shows the overall architecture of the system. For brevity, it does not show all possible links among different nodes. The B_i's at the same depth from $\mathcal{D}$ are grouped into one $level$. We adopt a distributed approach where we envision the system as a structured overlay point-to-point network. A key topological property that these overlays should possess is to have no cycles. Approaches have been proposed for CBPS systems (for example [16]) in which the topology resembles a global spanning tree or per-source trees. While such approaches guarantee cycle free topologies, they limit the potential for further scalability and efficiency as they do not take into account the current subscription patterns and frequency. Therefore we use the idea of multiple

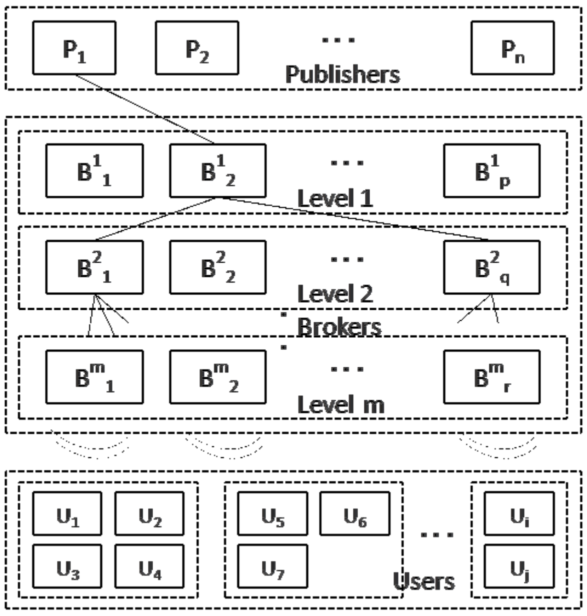

Fig. 1. The Overall Architecture of the System

levels of intermediate $\mathcal{B}$ to gain further advantage, by essentially creating multiple trees with roots being $\mathcal{D}$ and leaves being $\mathcal{U}$. The details about how we achieve this is discussed in a later section.

The scalability of an EDS largely depends on how the bandwidth utilization and the subscription routing tables scale, as the number of user groups (distinct subscriptions) increases. With $\mathcal{B}$ forming a DAG consisting of multiple trees from $\mathcal{P}$ to $\mathcal{U}$, we empirically show that our system satisfies the following fairly intuitive properties of a scalable EDS.

$\mathcal{SP}_1$ The total number of subscription groups monotonically decreases across *levels*[1] from bottom to top, that is, from $\mathcal{U}$ (which subscribe to the content) to $\mathcal{P}$ (which publish the content).

$\mathcal{SP}_2$ The total bandwidth utilization monotonically decreases across levels from bottom to top.

$\mathcal{SP}_3$ The bandwidth utilization monotonically decreases along any path from top to bottom.

In what follows, we use the term *subscription* to refer to one or more subtrees in the XML

[1]The B_i's at the same depth from $\mathcal{D}$ form a unique level in the DAG and each B_i belongs to only one level.

document to which U_i's or intermediate B_i's subscribe. We use the following terms in the rest of the paper.

Definition:[*Covering* and *Covered By* Subscriptions]

Let s_1 and s_2 be two subscriptions. We say that s_1 is *covering* s_2 if $s_2 \subseteq s_1$, that is, any node or edge in s_2 is also in s_1. Further, we say that s_2 is *covered by* s_1.

Definition:[*Identical* and *Distinct* Subscriptions]

Let s_1 and s_2 be two subscriptions. We say that s_1 and s_2 are *identical* if s_1 is *covering* s_2 and s_1 is *covered by* $s2$. By contrast, we say that s_1 and s_2 are *distinct* if neither s_1 is *covering* s_2 nor s_1 is *covered by* s_2.

III. Related Work

We briefly mentioned some broadly related work in the previous section. In this section, we critically evaluate the major research work closely related to ours.

Early CBPS systems model messages as *attribute value* pairs and a filter is essentially a logical formula containing constraints over the values of individual attributes (Gryphon [17], [18], Siena [16], [19], [20], [21]). While these content-based approaches are certainly more expressive than earlier subject-based systems [22], they are not suitable for hierarchical content, such as XML, filtering and distribution for they are designed to work with flat *attribute value* pairs.

With the popularity of XML as a standard data exchange format, there is a huge research base on XML filtering approaches which supports more expressive languages than tuple-based approaches. YFilter [5] which extends XFilter [1] to group FSM's into a Non-deterministic Finite Automata (NFA) exploiting the commonalities among path expressions. ONYX [23] leverages the YFilter to scale the content distribution. The approach by Chen et.al. [3] supports complex XPath expressions and builds an index structure called XTrie to perform efficient matching. XPush Machine [6] translates the entire XPath based queries into a single Deterministic Pushdown Automaton. Tian et al.[8] have proposed an XPath matching technique based on a relational database matching engine which supports a huge number of subscriptions without being constrained by the amount of memory available. All these approaches are based on subsets of XPath or XQuery and try to solve the problem of distributing different XML documents to different user groups based on their structure as well as content in some cases. The database and security communities have also carried out extensive research to securely distribute XML content

[2], [9], [10], [11], [12]. Most of these approaches have a considerable key management overhead, additional bandwidth overhead and indirectly leak sensitive infomation about the portions of the XML documents to which some U_i's $\in \mathcal{U}$ do not have access.

A major difference in our approach is we try to solve the problem of sending many different subdocuments of a few large documents to many different user groups while completely avoiding using XPaths in order to have better bounds on computational complexity. In fact, XPath based approaches are not directly suitable for sending incremental updates, where only updated portions are disseminated to save bandwidth, even if we deal only with relative XPath expressions. This is mainly due to the fact that XPath does not have a way to deal with subsets of XML documents especially when these expressions cross subset boundaries. Further, our selective dissemination approach minimizes indirect information leakage by not sending those portions of the XML documents to which U_i's do not have access.

IV. Our Approach

In this section, we describe the key techniques used in our approach using XML as the HDM and the terminologies used are consistent with the DOM specifications. First, we go through the process of annotation and encoding which is essential for structure-based extraction, matching, covering and merging. Then, we discuss how to handle subscriptions for XML documents. After that, we describe how to further compact subscription routing tables using covering and merging based routing while enforcing access control. Finally, we provide a brief overview of the brokering network.

A. Annotation and Encoding Scheme for Structure based Routing

The idea behind annotation is to associate enough information with each element of the XML document so that one can uniquely and efficiently identify each element in the document as well as efficiently perform subscription handling, matching and other operations. We associate two numerical values with each element e [24]: the post-order number (PON) and the lower bound (LB) of the tree rooted at e, $e(PON)$ and $e(LB)$ respectively. One may use other tree ordering schemes such as in-order or pre-order in order to achieve the same objective. The $e(PON)$ is the rank assigned by the post-order tree traversal in which each tree element e is traversed and assigned its post-order rank after its children are recursively traversed from left to right.The

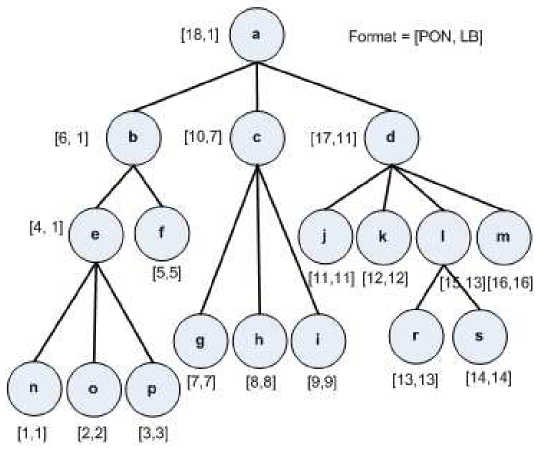

Fig. 2. A Simple Annotated Tree

$e(LB)$ is the lowest PON of the tree rooted at e. While we use integers for numbering through out this paper mainly for brevity, integers can be replaced by floating point numbers to prevent re-annotation when the structure is incrementally updated. Using floating point numbers also has potential security benefits. Having these two numbers associated with each node allows us to efficiently evaluate, among other things, subtree relationships.

The graph in Figure 2 shows an annotated document which we use as the running example through out this paper. Since annotations are associated only with element nodes, for brevity, the graph shows only such nodes and the corresponding edges. This annotation scheme also forms the basis for the tree data structure we propose for managing subscription routing tables.

The intuition behind the encoding scheme is to embed all non-element nodes except attribute nodes in XML documents within corresponding elements so that operations performed on the structure indirectly reflect on the actual content itself. Further, each element can be processed independently reaping huge benefits specially for incremental updates. The following formal specifications precisely capture our encoding scheme. We can view an XML document as a graph G = $\{$V, v, E, f, $g\}$ where:

V = V$_e$ $\cup$ V$_a$ $\cup$ V$_r$ where V$_e$ = $\{$x $|$ x is an element$\}$, V$_a$ = $\{$x $|$ x is an attribute$\}$, V$_r$ =

$\{$x | x is a node not in V$_e$ $\cup$ V$_a\}$ [2].

v = document root.

E = E$_e$ $\cup$ E$_a$ $\cup$ E$_r$ where E$_e$ = $\{$e | e is an edge representing an element-element connection or a link$\}$, E$_a$ = $\{$e | e is an edge representing an element-attribute connection$\}$, E$_r$ = $\{$e | e is an edge not in E$_e$ $\cup$ E$_a$ but starts from an element$\}$

f:E $\rightarrow$ L where L = $\{$l | l is an element node name or an attribute name or a pre-defined label$\}$, f is called the labeling function.

g:(V$_e$, i) $\rightarrow$ V$_{er}$ where g returns the ith child of V$_e$, V$_{er}$ = V$_e$ $\cup$ V$_r$

Let a_{PON} and a_{LB} be two attributes representing PON and LB values of a given element with the names APON and ALB respectively. After the encoding operation, we obtain a new XML document which corresponds to a graph G' = $\{$V', v, E', f', $g'\}$ where:

V' = V $\cup$ $\{$x | x is an attribute corresponding to a_{PON} or a_{LB} or an attribute embedding non-element nodes in V$_r$ $\}$ - V$_r$

E' = E $\cup$ $\{$x | x is an edge between $e \in V_e$ and a_{PON} or a_{LB} or new embedded attributes $\}$ - E$_r$

f':V' $\rightarrow$ L' where L' = L U $\{$ APON, ALB, labels of the embedded attributes $\}$

g':$\{$V$_e$, i$\}$ $\rightarrow$ V$_e$

Usually, XML documents have a good proportion of non-elements to elements. Therefore, this encoding process, in general, leads to further reduction of the number of nodes in the XML document, which, in turn, makes subsequent processing of these documents more efficient.

B. Subscription Handling

First we describe the Structure-based Subscription Language (SSL) we have devised for our approach.

Table 1 shows the SSL grammar. A `stree` denotes a tree rooted at an element e which we uniquely identify by the $(e(PON), e(LB))$ combination. If an XML document has at most n elements, there can be at most n $stree$'s and, hence, n subtrees. While the number of possible distinct subscriptions still remains 2^n, our grammar reduces the possible number of positive

[2]V$_r$ includes all non-element nodes except attribute nodes in an XML document

```
subscription → { subscription1 }
subscription1 → subscription1, subscription2
subscription1 → subscription2
subscription2 → subscription2 - stree
subscription2 → stree
stree → (digits, digits)
digit → [0-9]
digits → digit+
```

TABLE I

SSL GRAMMAR

combinations from 2^n to n where n is the number of elements in the XML document. This distinction is important since internal B_i's that do not interface any U_i's deal only with positive combinations. Such drastic reduction is one of the reasons for the efficiency of our approach while maintaining a level of expressiveness sufficient to cover practical subscription criteria. The intuitive semantic meaning behind our grammar is to grant positive authorization for a subtree along with negative authorizations for zero or more subtrees within the positive authorization. The SSL is fairly low level and designed to work efficiently among B_i's. We don't expect subscribers to be aware of SSL and they should be provided a high level interface to the SSL, the details of which is left to the extended version of the paper. The following example provide some applications of the SSL grammar.

Example 1: The subscription $\{(18,1) - (10,7)\}$ corresponds to whole XML document in Figure 2 except for the subtree rooted at $(10,7)$. The subscription $\{(6,1) - (4,1), (17,11)\}$ corresponds to the two subtrees of the XML document in Figure 2 rooted at $(6,1)$ (without the subtree rooted at $(4,1)$) and $(17,11)$ respectively.

Each D_i and each B_i maintain a subscription routing table to handle subscriptions efficiently. It is actually not a table as we find in many routing schemes but a tree data structure which we call an MXB tree. An MXB tree reflects the structure of the XML document. Each node in an MXB tree can have at most N_{mxb} children where N_{mxb} is the upper bound on the number of

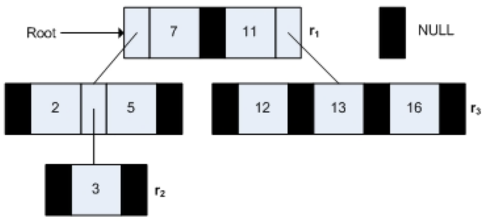

Fig. 3. The MXB Tree for Example 2

branches in the DOM tree corresponding to the XML document. If an element in the DOM tree has at most k subtrees, the corresponding node can have at most $2k-1$ entries, out of which $k-1$ are separation entries and k subtree pointers. An MXB tree can have at most H_{mxb} levels where H_{mxb} is the upper bound on the hierarchical depth in the corresponding DOM tree.

The complexity of an MXB tree depends on two factors. The upper bound on the branching factor N_{mxb} and the upper bound on the hierarchical depth H_{mxb}. These two factors, in turn, directly depend on the maximal XML document we can produce from a given XML Schema. It is reasonable to assume that each non-leaf XML element has at least two children and we call this the **minimum degree**, *t* (≥ 2) of the MXB tree. It is easy to show that the inequality $h \leq log_t(N_{mxb})$ holds for the MXB tree. The height of the tree grows in the order $O(log(N_{mxb}))$. Even though simple binary trees also have logarithmic growth, the base of the logarithm can be many times larger. Thus, MXB trees save a factor of about $O(log(t))$ over simple binary trees in the number of nodes traversed for most tree operations. We illustrate the MXB tree data structure through the following examples before we present detailed algorithms in another section.

Example 2: The MXB tree in figure 3 corresponds to the three simple subscriptions $\{(18,1)\}$ (i.e. the whole XML document), $\{(4,2)\}$ and $\{(17,11)\}$ made by r_1, r_2 and r_3 respectively.

We decide on splitting the entries of a node when there is at least one subscriber for that node or its descendant(s). For example, consider Figure 3. Nodes $(18,1)$, $(4,2)$ and $(17,11)$ are instantiated as they have one subscriber each. Node $(6,1)$ is instantiated as it is a successor of node $(4,2)$.

Example 3: Suppose that we get a new subscription $\{(10,7),\ (17,11)\}$ from r_4. As a result, the MXB tree in Figure 3 changes to the tree in Figure 4. Notice that r_4 is added against two

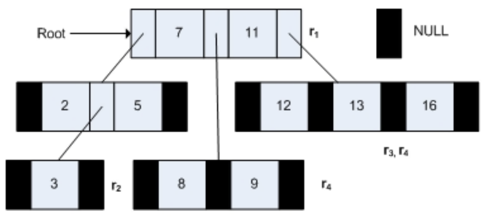

Fig. 4. The MXB Tree for Example 3

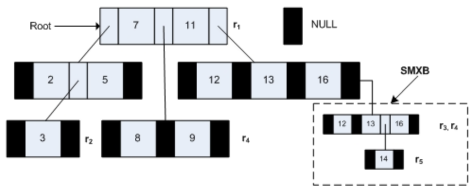

Fig. 5. The MXB Tree for Example 4

nodes in the MXB as it satisfies two subtrees.

So far, we represented only positive subscriptions. All internal B_i's, that do not interface any clients, maintain only positive subscriptions to reduce the complexity of subscription handling. Only external B_i's that interface $\mathcal{U}$ maintain a substructure to account for negative subscriptions. We trade the complexity of routing tables to false positives. It does not violate access control policies since the possible false positives traverse only within the broker network and are not propagated to $\mathcal{U}$. These substructures are also MXB trees rooted at each node. We refer to them as Sub-MXB's (SMXB for short). The following example illustrates the point.

Example 4: Suppose that we get a new subscription $\{(17, 11) - (15, 13)\}$ from r_5. As a result, the MXB tree in Figure 4 changes to the tree in Figure 5. Notice that r_3 and r_4 remains at the root of the SMXB while r_5 is pushed one level down.

C. Covering and Merging Based Routing

The goal of covering based routing is to remove redundant subscriptions and make subscription routing tables compact. The goal of merging based routing is to combine subscriptions that do not fall under covering relationships to further compact subscription routing tables. Both techniques in turn greatly reduce the network traffic. Prior work on merging and covering has explored different techniques mostly for attribute-value based PS systems, but XML based data distribution systems have not systematically investigated on this topic.

1) Covering: We first describe the concept for subscriptions with one subtree, then we generalize it to subscriptions with one or more subtrees. When a B_i receives a new subscription s, it checks if any existing subscription s_e covers s. If such a subscription exists, the request is not propagated further upwards and the requesting B_i is indexed against the covering subset of the exiting subscription s_e. Let $s(PON)$ be the PON of root element of the tree corresponding to the subscription s. The following simple implication shows the covering requirement:

$$(s_e(LB) \leq s(PON) \leq s_e(PON)) \Rightarrow s \subseteq s_e$$

The above relationship can be efficiently identified from the MXB tree data structure. If the root node has a subscription, it covers any incoming subscription; otherwise, we follow each entry such that the two neighboring keys cover the range ([$s(PON)$-$s(LB)$]) of the new subscription request. If there is a subscription along any of the nodes along the path, before we reach a NULL or an unqualified entry, that is, the subtree under the entry does not cover the range, the *covering* relationship holds. We can easily detect *identical* covering relationship during the same procedure. If the covering relationship holds and the value of the right key of the entry led to the matching node in the MXB tree is (1 + $s(PON)$), then we have an identical covering. This operation can be performed with a logarithmic complexity in the total number of distinct subscriptions we currently have at the given B_i.

If the covering relationship does not hold, the B_i checks for *covered by* relationships. Existing approaches for checking those two relationships on attribute-value based PS systems require at least two protocol rounds. By contrast, our approach detects either covering or covered by relationship in one protocol round. The MXB tree node for which the algorithm detects that the covering relationship is not satisfied, becomes the reference point for the covered by relationship or a new subscription altogether. In either case, the B_i itself cannot satisfy the subscription request and needs to propagate the request upwards. When a subscription request has multiple subtree requests, we simply break it into multiple subscriptions each having a single subtree. These broken down subscriptions may be good candidates for merging as described below.

2) Merging: We can view merging as an extended optimization to covering in order to further reduce the size and complexity of routing tables. Subscriptions that are not under covering relationships become the candidates for merging. As in the literature, we identify two types of merging; *perfect* merging and *imperfect* merging.

Let $s_1, s_2, ..., s_n$ be the candidate subscriptions and s_m be the subscription formed by merging $s_1, s_2, ..., s_n$.

A merging is *perfect* if $(s_1 \cup s_2 \cup ... \cup s_n) = s_m$, that is, the number of nodes and edges before and after merging is constant, and *imperfect* if $(s_1 \cup s_2 \cup ... \cup s_n) \subset s_m$, that is, the subscription resulting from the merging has extra nodes and/or edges compared to the candidate subscriptions. Since merging is performed only on those subscriptions that do not fall into any covering relationships, there is opportunity only for imperfect merging operations. However, resulting subtrees may create oppertunity for subsequent covering relationships. While imperfect merging reduce the complexity of the routing table, it may introduce false positives to the distribution network. Therefore, we need to quantify the imperfection and decide on when to perform imperfect merging. In order to respect access control policies, we perform merging only among internal B_i's.

Aggressive merging, in which each incoming subscription, that does not satisfy any covering relationship, triggers the merging procedure, would generate the most compact routing table. However, such a strategy incurs tremendous amount of overhead. Therefore we opt for performing merging only periodically to strike a balance between processing overhead and routing table complexity.

We quantify imperfect merging of subscriptions based on how similar one subscription is to other existing subscriptions. We introduce the *Subscription Distance (SD)* measurement to quantify the similarity. SD consists of the following two components:

1) *Upward Distance (d)*: The number of edges joining the roots of the two subtrees. If d is high, there is a good chance that these two subscriptions are not good candidates for merging.
2) *Additional Element Ratio (r)*: The number of additional elements that are included as a result of the merge compared to the total number of elements in the document. The lower the r is, the better are the candidates for merging.

$$SD = \{d, r\}$$

If $SD.d$ (≥ 2) is less than an empirically defined threshold d_t, we check if $SD.r$ ($\geq 1/n$ where n is the total number of elements in the document) is less than the threshold r_t to make the merging decision. It should be noted that imperfect merging is an iterative process and in between every merging iteration we perform covering.

D. Brokering Network

As introduced in the background section, the brokering network is a structured overlay point-to-point network that routes subscription requests from $\mathcal{U}$ towards $\mathcal{D}$ and document updates from $\mathcal{D}$ to $\mathcal{U}$. Routing paths constructed from $\mathcal{D}$ to $\mathcal{U}$ should be cycle free in order to have a correctly functioning system. As a consequence, it is the usual practice to construct a global spanning tree or per-source trees and route messages over the overlay network [16]. However, such schemes limit the ability to take advantage of related subscriptions or load balance subscriptions. Therefore, we take a different approach to dynamically decide the path between $\mathcal{U}$ and $\mathcal{P}$ by essentially creating multiple trees based on configurable policies.

We group $\mathcal{B}$ into levels. These levels are ranked from 1 to n. Thus, each broker is assigned a rank. Further, no broker is assigned to two or more levels. Level 1 B_i's are the closest to $\mathcal{P}$ and level n B_i's are the closest to $\mathcal{U}$. One may use different degree of connectivity among B_i's in two adjacent levels by changing configuration parameters. The higher the connectivity, the lower is the contention for B_i's. In order to prevent cycles, we have introduced the constraint that subscription requests are propagated from one B_i to another whose ranks are strictly decreasing. As a consequence, document updates are always routed from one B_i to another whose ranks are strictly increasing.

Subscription authorization is orthogonal to the work presented in this paper. We assume that B_i's can authorize the subscription requests made by U_i's before applying our approach.

V. Protocols and Algorithms

In this section we discuss some of the key protocols and algorithms used in our approach.

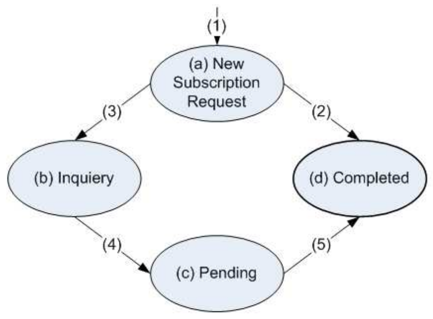

Fig. 6. Subscription States

A. *Network Start-up Protocol*

$\mathcal{B}$ and $\mathcal{D}$ form a structured P2P overlay network. New B_i's and/or P_i's can be added to the brokering network with minimal changes to the existing network configuration while the system is up and running. Each B_i/P_i is associated with a configuration file loaded at start up.

Correct and up-to-date neighboring node information is vital to route subscription messages upwards and update messages downwards. In order to have a highly dynamic environment, each node in the overlay identifies its neighboring nodes listed in the configuration file only at run time by message passing. When a B_i boots up, it informs about its presence to others so that relevant B_i's and P_i's can update their neighboring node information whereas the rest just ignore the notification. A reverse notification mechanism is available for newly started B_i's and P_i's to acquire information of their own neighboring nodes. Default configuration files are set up in such a way that B_i's have multiple paths to communicate with P_i's.

B. *Subscription Protocol*

We first look at the possible subscription states that the protocol can be at any B_i at any given time.

As shown in Figure 6, a subscription can be in four possible states. The following events trigger the state transitions and the order corresponds to the indices in Figure 6.

1) B_i receives a new subscription.

2) If the access control restrictions are not satisfied, B_i rejects the subscription. Otherwise, if B_i determines that its existing subscriptions cover the new subscription, B_i accepts the subscription without forwarding the request upwards.
3) If B_i determines that its existing subscriptions do not cover the new subscription request, it inquires from all its upward neighboring B_i's the availability of the subtree(s) corresponding to the new subscription. Depending on the *Subscription Propagation Policy* (SPP, described later in this section), it waits for all or some responses from its neighbors.
4) Once the inquiry protocol is completed, B_i makes a subscription request to the eligible neighboring B_i on behalf of the recipient (either a B_i or a U_i) and waits for the subscription reply.
5) Once the subscription request is received, B_i sends the response back to the recipient.

It should be noted that there can be multiple instances of the subscription protocol concurrently running. We associate a unique identifier with each subscription protocol in order to prevent any ambiguities among concurrent protocol rounds.

When the subscription requests are propagated upwards, an interesting issue is how to identify which neighboring B_i to choose from possibly many candidates. Such selection is carried out according to SPP's. Practically, it is difficult to devise a SPP that is optimal in all situations. Therefore, the system should be flexible enough to have different policies. The effectiveness of the covering based routing depends on the policy. In our approach we introduce the following two policies, but our approach can easily accommodate any new policy with minimal changes to the existing implementation.

- **First Fit** - After making an inquiry to all its neighbors, B_i selects the first response received and discards the rest. The intuition behind this approach is that the neighbor that responds first is less likely to be loaded compared to other neighbors and hence its selection may lead to better load distribution.
- **Best Fit** - After making an inquiry to all its neighbors, B_i waits until all or a majority of neighbors respond and selects the neighbor with the minimal *upward distance*. The intuition behind this approach is that the higher the similarity, the higher is the opportunity for covering and merging, thus leading to compact subscription routing tables.

If a B_i fails, all affected B_i's one level higher than the failed B_i get notified of the failure. The affected B_i's find different neighbors for those subscriptions that originally channeled through the failed B_i. This process is transparent to $\mathcal{U}$ unless the failed B_i is an interfacing broker. We assume that no two adjacent B_i's in any routing path fail at the same time. Handling such failures is left as future work.

C. Subscription Handling Algorithms

In what follows, we briefly discuss the three most important operations of MXB trees, that is, route, subscribe and unsubscribe in this sections.

The following conventions are used to describe the algorithms. A non-leaf node with m entries ($E_1 < E_2 <$ ... $< E_m$) contains $m + 1$ pointers (P_0, P_1, ..., P_m) to children. Pointer P_i points to a subtree whose PON values of each element are such that $E_i \leq PON < E_{i+1}$ where E_i is the value of the i^{th} entry of the node. As special cases, P_0 points to a subtree whose all PON values are less than E_1 and P_m points to a subtree whose all PON values are greater than or equal to E_m. N is the node of the MXB tree under consideration and $N(R)$ is the set consisting of all the recipients of the subtree rooted at N.

*a) **Algorithm 1: Route**:* It determines all recipients for whom a given subtree or a part of it should be sent. In other words, it is the *matching* algorithm that matches incoming XML document updates with subscriptions. We assume that the readers familiar with tree based algorithms will find it fairly easy to understand the intuition behind the algorithm.

*b) **Algorithm 2: Subscribe**:* We presented the intuition behind this algorithm in an earlier section. It takes the PON of the subtree to which a recipient subscribes and finds the appropriate node in the MXB tree (one or more nodes may be created on the fly if they are not instantiated yet) and inserts the recipient.

*c) **Algorithm 3: Unsubscribe**:* It takes the PON of the subtree from which the recipient r wants to unsubscribe from and finds the entry r and removes it. We remove the MXB node if it does not have any pointers going out of it as a result of removing a recipient, possibly resulting in a cascading delete. Due to the space constraint we have left the detailed algorithm for Unsubscribe and the analysis to the extended version of the paper except for mentioning that these algorithms have logarithmic complexity in the total number of distinct subscriptions.

Algorithm 1 Route(The root of the subtree to be sent: PON, The MXB tree node: N, Recipients:R)

```
1: if ($PON < E_1$ of $N$) then
2:     $R = R \cup N(R)$
3:     if ($PON$ == $E_1$ - 1) then
4:         {subtree matches exactly with the node}
5:         Send the subtree rooted at PON to all members of R and $(*P_0)(R)$
6:         $R = \Phi$
7:         $N$ = The node in MXB pointed to by $P_0$ of $N$
8:         for each subtree of $PON$: $PON_{sub}$ do
9:             Route($PON_{sub}$, $N$, $R$)
10:        end for
11:    else
12:        $N$ = The node in MXB pointed to by $P_0$ of $N$
13:        Route($PON$, $N$, $R$)
14:    end if
15: else if ($PON \geq E_m$) then
16:    $R = R \cup N(R)$
17:    if ($PON$ == $MAX$ at $N$) then
18:        {Code omitted - similar to the above if case, except for $P_m$ in place of $P_0$}
19:    else
20:        Route($PON$, $*P_m$, $R$)
21:    end if
22: else
23:    Find $i$ such that $E_i$ of $N \leq PON < E_{i+1}$ of $N$
24:    $R = R \cup N(R)$
25:    if ($PON$ == $E_{i+1}$ - 1) then
26:        {Code omitted - similar to the above if case, except for $P_i$ in place of $P_0$}
27:    else
28:        Route($PON$, $*P_i$, $R$)
29:    end if
30: end if
```

VI. Evaluation

In this section we provide major experimental results. The goals of the evaluation are of two fold:

1) To show that our approach satisfies the three properties mentioned earlier.
2) To compare the two *policies*, *First Fit* and *Best Fit*.

All the experiments were carried out with synthetic XML documents. We have developed flexible tools to generate XML documents with different patterns (number of branches, depth, etc.).

Algorithm 2 Subscribe(The root of the subtree to be subscribed to: PON, The MXB Tree Node: N, Recipient: r)

```
1: if ($PON \geq E_m$) then
2:     if ($PON$ == $MAX$ at $N$) then
3:         $N(R)$ = $N(R) \cup \{r\}$
4:     else
5:         Subscribe($PON$, $*P_m$, $r$)
6:     end if
7: else
8:     {Declare $P$, $E_L$, $E_U$}
9:     if ($PON < E_1$) then
10:        $P$ = $P_0$; $E_L$ = 1; $E_U$ = $E_1$
11:    else
12:        Find $i$ such that $E_i \leq PON < E_{i+1}$
13:        $P$ = $P_i$; $E_L$ = $E_i$; $E_U$ = $E_{i+1}$
14:    end if
15:    if ($PON$ == $E_L$ - 1) then
16:        if $P$ is $NULL$ then
17:            Create $*P$ based on the subtree ($E_L$, $E_U$ - 1)
18:        end if
19:        $*P(R)$ = $*P(R) \cup \{r\}$
20:    else
21:        Subscribe($PON$, $*P$, $r$)
22:    end if
23: end if
```

We assigned each element the same amount of text data so that number of nodes under a subtree is proportional to the bandwidth utilization when that subtree is sent. All the XML documents used have over 1000 elements. In each experiment, we formed 250 distinct subscription groups (if there are 100 U_i's on average per group, it could be interpreted as 25,000 subscriptions). The subtrees corresponding to these subscription groups are chosen randomly.

The brokering network was set up with five levels with each level having increasing number of *Content Brokers* as the rank of the level increases. B_i's were configured in such a way that each B_i knows all the brokers in the two adjacent levels.

These experiments (Figures 7, 8 and 9) were separately carried out for each *subscription propagation policy* multiple times and the average was taken. Since, to the best of our knowledge, this is the first implementation of PS systems exploiting structural properties, the experimental results are used mainly as a proof of the rate at which our approach scales.

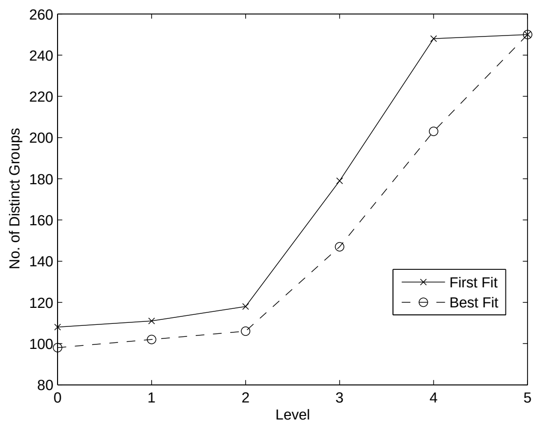

Fig. 7. No. of Distinct Groups vs. Levels

Figure 7 shows the number of distinct user groups formed at each level of the brokering network. As we go from bottom B_i's (i.e. level 5) to top B_i's, the number of distinct subscription groups substantially decreases. In other words, the subscriptions are fine-grained towards the bottom of the tree and coarse-grained towards the top of the tree. This experiment shows our approach satisfies the first property, $\mathcal{SP}_1$. This observation is consistent with the fact that as the subscriptions propagate from bottom to top, *covering* relationships combine many smaller subscriptions into larger subscriptions. The steep descent as the rank of the levels decreases suggests that our covering and merging based routing techniques are effective. The number of subscription groups reduces approximately by 12% of the initial count as the rank decreases. *Best Fit* criterion has a higher reduction factor compared to *First Fit* criterion. This observation is consistent with the fact that *Best Fit* criterion has a higher probability of creating covering and merging opportunities than *First Fit* criterion.

Figure 8 shows the bandwidth utilization at each level of the brokering network. As we go from bottom B_i's to top B_i's the bandwidth utilization across levels also drastically decreases. This experiment shows our approach satisfies the second property, $\mathcal{SP}_2$. This observation is consistent with the fact that bottom B_i's have many small subscriptions whereas top B_i's have only a few large subscriptions. The amount of bandwidth utilized reduces approximately by 14% of the initial value as the rank decreases. Due to the same reasons mentioned for the first experiment, *Best Fit* criterion has better bandwidth utilization than *First Fit* criterion.

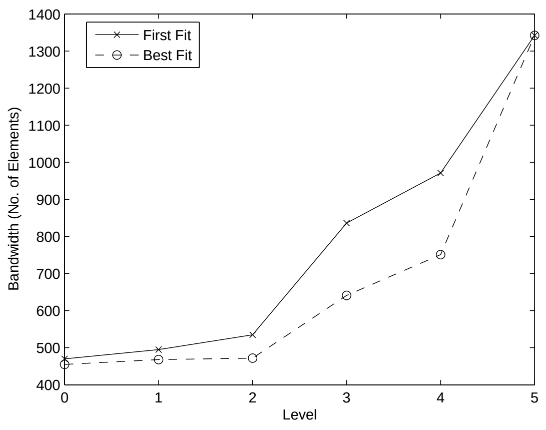

Fig. 8. Bandwidth Utilization vs. Levels

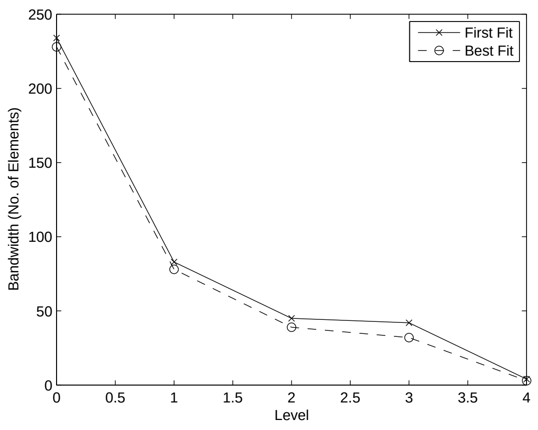

Fig. 9. Bandwidth Along a Path

Figure 9 shows the average bandwidth utilization along any path from top B_i's to bottom B_i's. The bandwidth along any path in the tree decreases from top B_i's to bottom B_i's. This experiment shows our approach satisfies the third property $\mathcal{SP}_3$. As the subscription granularity and branching increases towards bottom B_i's, one can expect to have the bandwidth split among multiple branches leading to the above observation.

As we can see from these experiments, while different subscription methods provide different rate of convergence, they all satisfy the three properties. Standard tree topologies suffer from a communication bottleneck towards the root of the tree as there are fewer links. To overcome this deficiency, our approach reduces the load towards the root of the trees to compensate for the communication bottleneck. We can infer from these experiments that, as the number of levels in the tree is increased, the scalability of the system dramatically increases by creating more opportunities for covering and merging relationships. Further, with more levels, subscription groups are also divided among increased number of leaf B_i's further boosting load handling capability.

VII. Discussion

In this section we look into additional requirements that may arise in other practical PS systems and discuss how our approach can be used to address them.

A. Enforcing Integrity and Confidentiality

Data security is particularly important when a third-party brokering network is utilized to distribute content [25]. For commercial applications, confidentiality is often a key requirement and for other applications integrity may be even more important than confidentiality.

Integrity enforcement equips subscribers with enough data to make sure that any partial XML document sent is not tampered by any unauthorized party. With hierarchically-structured data such as XML documents, we need to enforce both content integrity and structural integrity. Since our encoding scheme makes each XML element self-contained and the access control granularity is the XML element, one possible way to enforce integrity is to associate each XML element with sufficient meta data to uniquely identify parent child relationship and apply any existing digital signature scheme to each XML element.

Performing access control on published documents before sending them to subscribers, is only one aspect of confidentiality. Concealment of data from intermediaries including content brokers and/or hiding the existence of unauthorized parts of the document (thus, minimizing indirect information leakage) could also be important when distributing highly confidential documents. Similar to integrity enforcement, we need to hide not only their content but also the structure. At the granularity of encoded XML elements, one way to hide the content is to encrypt each XML element in order to create a clone of the original XML document replacing the elements in the clear with encrypted elements. Two possible strategies for hiding the structure in our approach is to remove annotation information at interfacing content brokers or introduce an annotation scheme which makes inferring of the structure difficult but still demonstrates the properties of PON's. Secure random numbers or numbers encrypted with order preserving schemes [26] can be used for this purpose.

B. Publishing Incremental Updates

For certain types of contents, not all sections of XML documents change with the same frequency. In such scenarios, publishing only the updated sections can improve bandwidth utilization and result in reduced processing overhead.

Most of the existing approaches including those based on XPath or XQuery, view XML document as a whole, thus making incremental publishing more difficult. In our approach, any type of processing, such as integrity enforcement and verification, encryption and decryption, on XML documents can be carried out independently at the granularity of XML elements. Therefore, our approach inherently supports publishing incremental updates.

C. Content-based Filering

Our approach provides content-based filtering to some extent. However, if further expressivenss is required, our approach can be extended to support filtering XML documents based on the actual content itself. With access controlled documents and content based filtering, inferencing of restricted information is a major issue. A key insight we derive from prior work is that we can avoid such issues by making queries to apply filters only to those parts of the XML documents recipients have access to. A two step process, where we first isolate content based on the structure of documents and then apply a suitable technique to the isolated content in order to perform the final filtering, could realize this goal. Our structure based filtering can be the basis for the first step of this sand-boxing approach.

VIII. Conclusion and Future work

We proposed an efficient and scalable approach to distribute different subtrees of possibly large documents to different user groups by exploiting the hierarchical structure of those documents. Our approach is based on the use of a simple yet effective subscription language and a novel tree data structure for efficiently constructing routing tables. We presented a further optimization based on covering and merging based routing to reduce the load towards the root of the distribution trees leading to a scalable system. We also introduced a policy-based dynamic subscription routing protocol that increases the opportunity for covering and merging relationships. The experimental results show that our claims are correct. The ability to efficiently deal with partial documents is an additional advantage of our approach.

We plan to extend this work in the following directions. A different type of XML PS systems looks at the actual content to enforce access control policies and make routing decisions. We believe that the approach presented in this work can be extended to analyze both the structure and the content of XML documents in order to deal with such systems. Extending the brokering network to handle byzantine failures is another direction we are planning to work on. We also plan to conduct further experiments including the trade-off between false positives (additional bandwidth) and matching efficiency.

References

[1] M. Altinel and M. Franklin, “Efficient filtering of XML documents for selective dissemination of information,” in *VLDB ’00: Proceedings of the 26th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 53–64.

[2] L. Opyrchal and A. Prakash, “Secure distribution of events in content-based publish subscribe systems,” in *SSYM’01: Proceedings of the 10th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2001, pp. 21–21.

[3] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi, “Efficient filtering of XML documents with xpath expressions,” *The VLDB Journal*, vol. 11, no. 4, pp. 354–379, 2002.

[4] L. Lakshmanan and S. Parthasarathy, “On efficient matching of streaming XML documents and queries,” in *EDBT ’02: Proceedings of the 8th International Conference on Extending Database Technology*. London, UK: Springer-Verlag, 2002, pp. 142–160.

[5] Y. Diao, M. Altinel, M. Franklin, H. Zhang, and P. Fischer, “Path sharing and predicate evaluation for high-performance XML filtering,” *ACM Trans. Database Syst.*, vol. 28, no. 4, pp. 467–516, 2003.

[6] A. Gupta and D. Suciu, “Stream processing of XPath queries with predicates,” in *SIGMOD ’03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2003, pp. 419–430.

[7] T. Green, G. Miklau, M. Onizuka, and D. Suciu, “Processing XML streams with deterministic automata,” in *ICDT ’03: Proceedings of the 9th International Conference on Database Theory*. London, UK: Springer-Verlag, 2002, pp. 173–189.

[8] F. Tian, B. Reinwald, H. Pirahesh, T. Mayr, and J. Myllymaki, “Implementing a scalable XML publish/subscribe system using relational database systems,” in *SIGMOD ’04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2004, pp. 479–490.

[9] E. Bertino, S. Castano, and E. Ferrari, “On specifying security policies for web documents with an XML-based language,” in *SACMAT ’01: Proceedings of the sixth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2001, pp. 57–65.

[10] G. Miklau and D. Suciu, “Controlling access to published data using cryptography,” in *vldb’2003: Proceedings of the 29th international conference on Very large data bases*. VLDB Endowment, 2003, pp. 898–909.

[11] E. Bertino, B. Carminati, E. Ferrari, B. Thuraisingham, and A. Gupta, “Selective and authentic third-party distribution of XML documents,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 10, pp. 1263–1278, Oct. 2004.

[12] B. Carminati, E. Ferrari, and E. Bertino, “Securing XML data in third-party distribution systems,” in *CIKM ’05: Proceedings of the 14th ACM international conference on Information and knowledge management*. New York, NY, USA: ACM, 2005, pp. 99–106.

[13] A. Kundu and E. Bertino, “A new model for secure dissemination of XML content,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 3, pp. 292–301, May 2008.

[14] G. Gottlob, C. Koch, and R. Pichler, “Efficient algorithms for processing XPath queries,” *ACM Trans. Database Syst.*, vol. 30, no. 2, pp. 444–491, 2005.

[15] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, “The many faces of publish/subscribe,” *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[16] A. Carzaniga, M. Rutherford, and A. Wolf, “A routing scheme for content-based networking,” in *INFOCOM*, 2004, pp. 918–928.

[17] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra, “Matching events in a content-based subscription system,” in *PODC ’99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 1999, pp. 53–61.

[18] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman, “An efficient multicast protocol for content-based publish-subscribe systems,” *icdcs*, vol. 00, p. 0262, 1999.

[19] A. Carzaniga and A. Wolf, “Forwarding in a content-based network,” in *SIGCOMM ’03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2003, pp. 163–174.

[20] F. Fabret, H. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha, “Filtering algorithms and implementation for very fast publish/subscribe systems,” *SIGMOD Rec.*, vol. 30, no. 2, pp. 115–126, 2001.

[21] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman, “Exploiting ip multicast in content-based publish-subscribe systems,” in *Middleware ’00: IFIP/ACM International Conference on Distributed systems platforms*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2000, pp. 185–207.

[22] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen, “The information bus: an architecture for extensible distributed systems,” *SIGOPS Oper. Syst. Rev.*, vol. 27, no. 5, pp. 58–68, 1993.

[23] Y. Diao, S. Rizvi, and M. Franklin, “Towards an internet-scale XML dissemination service,” in *VLDB ’04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004, pp. 612–623.

[24] P. Dietz, “Maintaining order in a linked list,” in *STOC ’82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1982, pp. 122–127.

[25] C. Wang, A. Carzaniga, D. Evans, and A. Wolf, “Security issues and requirements for internet-scale publish-subscribe systems,” Jan. 2002, pp. 3940–3947.

[26] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, “Order preserving encryption for numeric data,” in *SIGMOD ’04:*

Proceedings of the 2004 ACM SIGMOD international conference on Management of data. New York, NY, USA: ACM, 2004, pp. 563–574.