Routing of XML and XPath Queries in Data Dissemination Networks

Guoli Li Shuang Hou Hans-Arno Jacobsen
Middleware Systems Research Group, University of Toronto
{gli,shou}@cs.toronto.edu jacobsen@eecg.toronto.edu

Abstract

XML-based data dissemination networks are rapidly gaining momentum. In these networks XML content is routed from data producers to data consumers throughout an overlay network of content-based routers. Routing decisions are based on XPath expressions (XPEs) stored at each router. To enable efficient routing, while keeping the routing state small, we introduce an advertisement-based routing algorithm for XML content, present a novel data structure for managing XPEs, especially apt for the hierarchical nature of XPEs and XML, and develop several optimizations for reducing the number of XPEs required to manage the routing state. The experimental evaluation shows that our algorithms and optimizations reduce the routing table size by up to 90%, improve the routing time by roughly 85%, and reduce overall network traffic by about 35%. Experiments running on PlanetLab show the scalability of our approach.

1. Introduction

Over the past decade, XML has rapidly evolved as the standard for data representation and exchange. XML marked-up message traffic in intranets and on the Internet range from insurance claims, health-care requests, corporate memos, online ads to news items and entertainment information. The standardization of the mark-up language, the wide range of related standards, and the wide-spread adoption of this technology are further amplifying the network externalities created by this technology.

XML-based data dissemination networks are starting to become a reality. In a dissemination network, data, marked-up in XML, is routed based on filter expressions stored at intermediate nodes that indicate where the XML document is to be routed to. Filter expressions, often expressed as XPath expressions (XPEs), are submitted by data consumers who express interest in receiving certain kinds of documents. For instance, a globally operating insurance company with many branch offices distributed world-wide is linked by an overlay network of content-based routers that comprise the XML dissemination network. An insurance claim, an insurance bid, or a request for proposal can be submitted anywhere into the overlay network (e.g., by a third party insurance broker or an online client) and be routed toward a currently online, specific expert employee, speaking the same language as the requester. Note, the latter constraints are expressed as XPE filter expressions against which the XML document is evaluated in transit. This design fully decouples information requesters and information providers, avoids a single point of control and a single point of failure, and increases scalability due to decentralization and distribution.

This paper addresses the XML/XPath routing problem. More specifically, this paper focuses on the problem of efficiently routing an XML document emitted from a data producer at one point in the network to a set of data consumers located anywhere throughout the network. Prior to receiving XML documents, consumers must have expressed interest in receiving XML documents by registering XPEs with the network. This problem statement is akin to the well-known publish/subscribe matching problem. However, the main difference here is that in the case of data dissemination networks there exists no one single centralized publish/subscribe system, but a network of content-based routers (i.e., a network or federation of publish/subscribe systems.) In the dissemination network, XML documents are routed based on their content and not based on IP address information, which is, due to the completely decoupled design, not available – all routing decisions are exclusively based on content information. Figure 1 provides an overview of the dissemination network this paper assumes. In the overlay network depicted in Figure 1 each content-based router, referred to as broker, only knows its neighbors (i.e., in terms of IP network address information.) However, none of the clients – neither data producers nor data consumers know about each other or about the network topology, except the router they connect to.

In the context of XML-based data dissemination, one of the main challenges is the ability to efficiently deliver relevant XML documents to a large and dynamically changing group of consumers. Centralized XML filter-

1063-6927/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDCS.2008.31

ing [3, 10, 7] and distributed query-based XML retrieval approaches [6, 19, 18] have found wide-spread interest, but do not address the distributed, content-based routing problem articulated above and addressed in this paper. ONYX is a query-based XML retrieval approach [11] for XML data dissemination in distributed environments. It is closely related to our approach, but complementary in objectives. ONYX aims at reducing the XML message size. ONYX achieves this through only disseminating parts of an XML message selected by subscriber queries. Several techniques are presented and evaluated to achieve this objective. In the context of our work, subscriber queries are meant to select messages that match the query, however, the subscriber requires the full message and not just parts of it. It is therefore difficult to quantitatively compare both approaches. Content-based routing [5, 24, 9] in distributed publish/subscribe architectures, have been studied for non-XML-based data. Their operational model assumes sets of attribute-value pairs joined by Boolean operators. It is not at all obvious how to extend these approaches for semi-structured data, especially due to the hierarchical data model of XML.

Our own prior research on content-based routing [21, 14, 20, 8] develops architectures, algorithms and protocols for content-based routing of non-XML based data. It is a non-trivial problem, as we argue in this paper, to extend these approaches to the hierarchical structure of XML. Our prior work on developing efficient matching algorithms for XML [16] addresses part of the problem, but does not address the important problem of efficiently computing routing decisions, inducing advertisements from XML DTDs, and determining covering and merging relations, which is the focus of this research. This paper combined with our earlier work on XML matching [16] comprises all components required to build an efficient content-based router for an XML data dissemination network. The research presented in this paper complements our PADRES content-based publish/subscribe system effort [14, 1]. PADRES has to date not investigated the routing of XML content against XPEs.

In this paper, we develop algorithms for dissemination of XML data throughout a network of content-based routers towards data consumers who have specified their interests through XPEs. Our contributions are: first, we adapt the use of advertisements to optimize data dissemination. While this idea is common in the publish/subscribe literature [5, 24, 9], it is not clear how to extend the concepts to the data model of XML. We demonstrate how to use the XML Document Type Definition (DTD) to generate advertisements about the information a data producer is going to publish. We distinguish between a non-recursive and a recursive case depending on the DTD defining the data emitting source. We then develop advertisement-based routing algorithms for both cases. Second, we propose a novel data structure to maintain XPEs by identifying the covering relations among them. We present covering algorithms for XPEs to reduce the routing table size stored at each router and speed up routing computation in the routers. Third, we present an optimization of merging similar XPEs to further reduce routing computation. Finally, we perform a detailed experimental evaluation of our approach on an overlay network comprised of 127 XML routers deployed over a cluster with 20 nodes and deployed on PlanetLab. Our experimental results demonstrate the effectiveness of the approach by reducing the routing table size by up to 90% and improving the routing time by roughly 85%.

2. Background

2.1. Content-based Routing

Content-based publish/subscribe systems [5, 24, 9, 14, 1] provide a flexible and extensible environment for information exchange. Publishers and subscribers are clients to the publish/subscribe system, are loosely coupled in space and time, and have no knowledge of each other. Messages in content-based publish/subscribe systems are routed based on their content rather than the IP address of their destinations. In order to handle a large amount of dynamic information and reduce the network traffic many optimization techniques, such as advertisements [5], covering technique and merging technique [9, 5, 24] have proven to offer significant benefits for non-XML based publish/subscribe systems. While conceptually, these ideas apply to XML-based data as well, it is not obvious how to apply these concepts to XML due to the structural complexity of XML data. These are the challenges addressed by this paper.

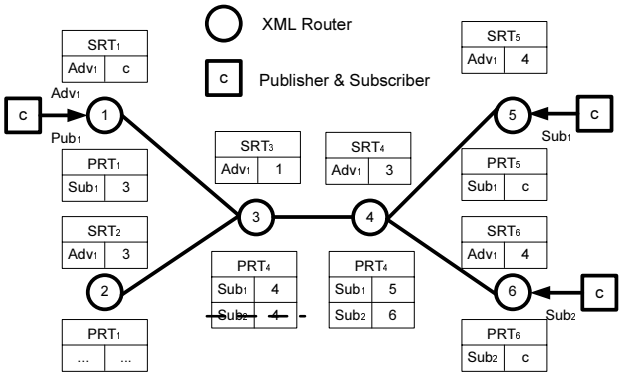


Figure 1. Content-based Routing

In advertisement-based publish/subscribe systems, advertisements are specifications of information that the publisher publishes in the future. Advertisements are flooded

in the publish/subscribe overlay. The common assumption is that the number of advertisements is much smaller than the number of subscriptions and publications. Advertisements are used to avoid broadcasting subscriptions in the network, so that subscriptions are only routed to the publishers who advertise what the subscribers are interested in. Subscriptions define filters to select publications of interest. Matching publications are delivered to subscribers along the paths built by subscriptions. Figure 1 shows a scenario for advertisement-based content-based routing. The subscription routing table (SRT) consisting of <advertisement, last-hop>-tuples stores advertisements in order to route subscriptions. Publications trace back along the path setup by subscriptions to interested subscribers. The publication routing table (PRT) maintains the path information. For example, in Figure 1, advertisement adv_1 is broadcast in the network, and is stored on each broker of the network with a different last hop. Consequently, subscriptions that match adv_1 are routed according to these last hops (e.g., sub_1 is routed along the link $5-4-3-1$). Note that the subscription sub_1 is not forwarded to Brokers 2 and 6, respectively, since adv_1 indicates that matching publications are from Broker 1. Therefore, publication pub_1 is routed along the reverse path $1-3-4-5$ to the subscriber. In the rest of this paper, we use the notations $P(s)$ and $P(a)$ to refer to the set of publications that match subscription s and advertisement a, respectively.

2.2. Covering and Merging

The goal of covering-based routing [5, 24] is to remove redundant subscriptions from the network in order to obtain a compact routing table and reduce the network traffic. In Figure 1, if subscription sub_1 covers subscription sub_2 at Broker 4, sub_2 is not forwarded to Broker 3. That is, we can safely remove sub_2 at Broker 3 obtaining a compacter routing table while maintaining the same information delivery behavior. All publications matching sub_2 must also match sub_1. A formal definition of the covering relation is as follows: A subscription sub_1 covers sub_2, if and only if, P(sub_1) $\supseteq$ P(sub_2), denoted as $sub_1 \sqsupseteq sub_2$. The covering relation defines a partial order on the set of all subscriptions with respect to $\sqsupseteq$. Since advertisements have the same format as subscriptions, the covering relations among advertisements can be defined in the same manner.

If two subscriptions are not in a covering relation, but their publication sets overlap with each other, the two subscriptions can be merged to a more general subscription, which covers the original subscriptions. Suppose subscription sub_m is a merger of sub_1 and sub_2, then we have P(sub_m) $\supseteq$ P(sub_1) $\cup$ P(sub_2). There are two kinds of mergers. If the publication set of the merger is exactly equal to the union of the publication set of the original subscriptions, the merger is a *perfect merger*; otherwise, if P(sub_m) $\supset$ P(sub_1) $\cup$ P(sub_2), it is an *imperfect merger*. After merging, only the merger is forwarded into the network. The merging technique [24] is used for further minimizing the routing table size, since the merger may introduce new covering relations among subscriptions. Covering and merging are complementary routing optimizations.

3. Advertisement-based Routing

Upon receiving a subscription, a broker matches the subscription against its advertisements. If there is an advertisement whose publication set overlaps that of the subscription, it means there is a *match* between the subscription and the advertisement. The broker then routes the subscription to the broker where the advertisement came from.

3.1. XML-based Advertisements

In the context of XML/XPath routing, advertisements are generated by exploiting DTD information. The purpose of a DTD is to define the legal building blocks of an XML document. The main building blocks of XML documents are elements surrounded by tags, e.g., $< root > ... < /root >$, where $root$ is the element name in the document. All elements appearing in the XML document must be defined in the corresponding DTD, which determines the structure of elements and their sequence in the document. In this paper, our discussion focuses on the main building block – elements. Our approach could be easily extended to element attributes and content [16], which we omit due to space limitations.

In this paper, we use the common interpretation of an XML document as a tree of nodes and consider each path from the root node to a leaf node. Thus, we decompose each XML document into a set of XML paths and each path is represented as $e = /t_1/t_2/.../t_n$, where t_i is the XML element name. These paths are extracted from the document before the publisher submits the document to the network. Thus, a publication routed in our system is actually an XML path annotated with a *pathId* and *docId*. This is transparent to publishers and subscribers who handle entire XML documents. Publishers submit entire XML documents, commonly referred to as publications, and subscribers submit XPath expressions (XPEs), commonly referred to as subscriptions. We use the terms XPE and subscription interchangeably in the rest of this paper.

We use an absolute XPath expression without //-operators as the format of advertisements in the context of XML/XPath data routing. Note that this is not a restriction of our subscription language. Advertisements are a system internal mechanism, which is not exposed to the application or to the user. An advertisement is described as

$a = /t_1/t_2/.../t_{n-1}/t_n$, where t_i can be either an element name or a wildcard, and a has the same length as the publication it advertises. In our approach, advertisements are derived from the DTD, since the DTD allows deriving all possible paths from the root to the leaves appearing in related XML documents.

We call an advertisement a *non-recursive* advertisement if it is extracted from a non-recursive DTD. The above advertisement a is an example of non-recursive advertisement. A DTD is recursive if it contains elements that are defined in terms of the elements themselves. The popular NITF DTD, often used for experimentation, is recursive. We call an advertisement a *recursive* advertisement if it is extracted from a recursive DTD. An advertisement may have multiple recursive parts that appear in sequence or are embedded in each other. We classify recursive advertisements into three categories as described below.

Simple-recursive advertisements: A simple-recursive advertisement has only one recursive pattern. The advertisement is described as $a = /t_1/t_2/.../t_{i-1}(/t_i/.../t_j)^+/.../t_n$, where the $+$ operator declares that elements $t_i, ..., t_j$ must occur one or more times in the advertisement. Note that this is not part of XPath syntax. Advertisements are only used within the system, so the extended XPath syntax has no effect on clients and applications. In the proposed algorithms, we use $a = a_1(a_2)^+a_3$ to simplify the expression, where a_k $(1 \leq k \leq 3)$ is a non-recursive advertisement.

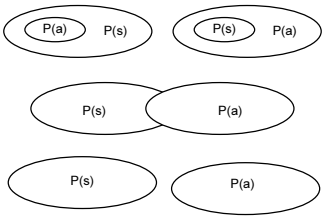
Series-recursive advertisements: A series-recursive advertisement includes more than one recursive pattern in sequence. For example, an advertisement containing two recursive patterns in sequence can be described as $a = /t_1/t_2/.../t_{i-1}(/t_i/.../t_j)^+/t_{j+1}.../t_{l-1}(/t_l/.../t_o)^+/.../t_n$, or as simplified expression with non-recursive advertisements, $a = a_1(a_2)^+a_3(a_4)^+a_5$.

Embedded-recursive advertisements: An embedded-recursive advertisement recursively embeds patterns in other patterns. A possible case is $a = /t_1/t_2/.../t_{i-1}(/t_i/.../t_{l-1}(/t_l/.../t_o)^+/.../t_j)^+/.../t_n$, or $a = a_1(a_2(a_3)^+a_4)^+a_5$. The embedded-recursive advertisement can be more complex.

More types of recursive advertisements can be easily defined based on the above three types of advertisements. We discuss the matching algorithms for non-recursive and recursive advertisements in Sections 3.2 and 3.3, respectively.

3.2. Non-recursive Advertisement

In this section, we discuss the algorithms for subscriptions and non-recursive advertisements matching in the context of XML/XPath. An advertisement a matches a subscription s if the publication sets $P(a)$ and $P(s)$ overlap, that is, $P(a) \bigcap P(s) \neq \Phi$. Figure 2(a) shows all possible relations between the two sets. To forward subscriptions, we need to identify the first two overlapping cases in Figure 2 (a). In this paper, we focus on the subscriptions including parent-child operator (/), wildcard operator (*), and ancestor-descendant operator (//). For other operators appearing in the subscription, such as attribute filters, our approach can be easily extended to support them through value comparison. We discuss the matching algorithm for the following three subscription cases.



Adv	Sub	Overlap
*	*	Y
*	t	Y
t	*	Y
t	t	Y
t_1	t_2	N

(b)

Figure 2. Adv. and Sub. Relations

Absolute simple XPEs: A simple XPE only contains parent-child and wildcard operators. We describe the matching algorithm for absolute XPEs (without //-operator) and advertisements (`AbsExprAndAdv`). For example, $s = /st_1/st_2/.../st_k$ and $a = /at_1/at_2/.../at_n$, where st_i is the i-th element of s and at_j is the j-th element of a. We use this notation in all algorithms in this paper. First, the algorithm does not have to be applied, if the given XPE is longer than the advertisement. This observation is exploited because the advertisement has the same length as its publications, and thus, publications in $P(a)$ do not match all the elements in the longer XPE. Next, the algorithm compares each pair of elements or wildcards in the advertisement and the subscription, according to the matching rules shown in Figure 2(b). It returns 0, if some pair does not overlap; otherwise it returns 1. For example, given $a = /b/*/*/c/c/d$ and $s = /*/c/*/b/c$, the algorithm returns 0, since the matching rules fail to satisfy for $i = 4$. As shown in Figure 2(b), the fifth row indicates that the advertisement includes an element c and the subscription includes an element b at the same position that do not overlap. That is publications matching the advertisement cannot match the subscription.

Relative simple XPEs: These expressions are similar to absolute simple XPEs except for the first operator, which cannot be a "/". That is the XPE is relative. The matching could start at any position of the advertisement because the subscription is relative. A naive matching algorithm for this case is repeatedly calling `AbsExprAndAdv`. In iteration i, the algorithm takes the subscription as an absolute one and starts the matching from the i-th position of the advertisement. We skip the details of the naive algorithm as it is

straightforward. The complexity of the naive algorithm is $O(n * k)$ where n is the length of the advertisement and k is the length of the subscription. We propose an optimized version of the matching algorithm for relative simple XPEs.

The matching algorithm for relative simple XPEs and advertisements (`RelExprAndAdv`) is a string matching problem [17]. We try to find the XPE, s, inside the advertisement, a, by starting at the first element of a that matches st_1 and continue (i.e., comparing to st_2 and so on) until we either complete the match or find a mismatch. In the latter case, we must go back to the place where we started. The difference between the traditional string matching problem and ours is that the wildcard "*" can match any element in our matching rules, as shown in Figure 2(b). To improve this algorithm, the KMP algorithm [17] is applied to reduce the number of comparisons to $O(n)$.

Descendant operators in XPEs: Descendant operators indicate that more than one element should appear in the matching advertisement. The matching algorithm for XPEs with descendant operators and advertisements (`DesExprAndAdv`) is based on the above XPE matching algorithms (i.e., `AbsExprAndAdv` and `RelExprAndAdv`). We split the XPE in maximal length sub-XPEs that do not contain any descendant operators, and match each sub-XPE against the advertisement with sequence comparison. For instance, given $a = /a/ * /e/ * /d/ * /c/b$ and $s = */a//d/ * /c//b$, the algorithm matches all sub-XPEs in s against a in order. It returns 1 because it finds each sub-XPE $*/a$, $d/ * /c$ and b matches different parts in a (e.g., $a/*$, $*/d/*$ and b).

3.3. Recursive Advertisement

Input: advertisement $a = a_1(a_2)^+a_3$, and subscription s ($a_k, 1 \leq k \leq 3$, is an advertisement)
Output: 1 if $P(a) \bigcap P(s) \neq \Phi$, 0 if $P(a) \bigcap P(s) = \Phi$
01: **If** $|s| \leq |a_1a_2|$ **then return** AbsExprAndAdv(a_1a_2, s)
02: **Else** *temp* $\leftarrow$ AbsExprAndAdv(a_1a_2, $/st_1/.../st_{|a_1a_2|}$)
03: **If** *temp* = 0 **then return** 0
04: **If** $|s| \leq |a_1a_2a_3|$ **then** $q \leftarrow 0$
05: **Else** $q \leftarrow Int((|s| - |a_1a_2a_3|)/|a_2|) + 1$
06: $p \leftarrow Int((|s| - |a_1a_2|)/|a_2|)$
07: **For** $c = q : p$ **do**
08: *temp* $\leftarrow$ AbsExprAndAdv(a_3, $/st_{c*|a_2|+|a_1a_2|+1}/.../st_{|s|}$)
09: **If** *temp* =1 **then return** 1
10: **If** $c = p$ **then**
temp $\leftarrow$ AbsExprAndAdv(a_2, $/st_{c*|a_2|+|a_1a_2|+1}/.../st_{|s|}$)
11: **Else** *temp* $\leftarrow$ AbsExprAndAdv(a_2, $/st_{c*|a_2|+|a_1a_2|+1}/.../st_{(c+1)*|a_2|+|a_1a_2|}$)
12: **If** *temp* =0 **then return** 0
13: **Return** 1

Figure 3. AbsExpr. & Simple RecAdv.

We focus on the matching of absolute XPEs and recursive advertisements. The matching of other types of XPEs and recursive advertisements can be implemented based on this algorithm. In Figure 3, the matching algorithm for absolute XPEs and simple recursive advertisements (`AbsExprAndSimRecAdv`) calls `AbsExprAndAdv` if the subscription is not longer than the recursive pattern (Line 1). If the subscription is longer, the algorithm estimates the maximum number that the recursive pattern would be repeated in the advertisement according to the length of both subscription and advertisement (Lines 4-6). Next, the algorithm tries all possible advertisements according to the maximum number of repeated recursive patterns (Lines 7-12). For example, given $a = /a/ * /c(/e/d)^+/ * /c/e$ and $s = /*/a/c/*/d/e/d/*$, first, the algorithm compares $/a/*/c/e/d$ in a with $/*/a/c/*/d$ in s, and computes $q = 0$ and $p = 1$ in Lines 4-6. Second, it supposes that the recursive pattern is repeated only once, compares $*/c/e$ in a with $e/d/*$ in s (Line 8) and fails to match. Next, it repeats the recursive part e/d twice, and continues the comparison (Line 11). Finally, it returns 1 (Line 9) if it finds a matches s with double recursive patterns in a. The complexity of the algorithm is $O(n^2)$, since it actually matches the subscription against each possible advertisement without recursive pattern. From a practical point of view, it is reasonable to limit the maximum nesting depth of items in a document, which would reduce the complexity of processing DTDs.

The matching algorithm for absolute XPE and series-recursive advertisements (`AbsExprAndSerRecAdv`), where $a = a_1(a_2)^+a_3(a_4)^+a_5$, is implemented by calling the algorithm from Figure 3 recursively. The matching determines how many times the first recursive pattern could be repeated, and calls the algorithm from Figure 3 repeatedly to try all possible advertisement formats. The matching of XPE and embedded recursive advertisements (`AbsExprAndEmbRecAdv`) is similar to `AbsExprAndSerRecAdv`. First, it determines how many times the outer recursive pattern could be repeated, and calls `AbsExprAndSerRecAdv` (not restricted to two recursive patterns) repeatedly.

4. Covering and Merging

In this section, first, we describe a novel data structure called *subscription tree* for maintaining subscriptions. The data structure captures the covering relations among subscriptions and speeds up the covering detection. Second, we present the covering algorithms for absolute simple XPEs, relative simple XPEs, and XPEs with descendant operators. Last, we explore the merging technique, and discuss the merging rules in the context of XPEs.

4.1 Subscription Tree

In covering-based routing, if an arriving subscription is covered by an existing subscription in the routing table, the new subscription is not forwarded to the next-hop broker. On the other hand, if the arriving subscription covers existing subscriptions, before it is forwarded, the broker needs to unsubscribe all the subscriptions that are covered by the new subscription. Therefore, the network traffic is reduced by removing the redundant subscriptions and the routing table in the next-hop broker is compacted.

At each broker, subscriptions are maintained in a tree data structure. The idea is to store the subscriptions according to the covering relations among them. A subscription at a node in the tree covers all subscriptions in its subtree. Since a covering relation defines a partial order among subscriptions, a tree data structure cannot capture all the covering relations. A subscription node can have only one parent in the tree, but it may be covered by several subscriptions. We allow each node having a set of *super pointers*, which indicate the covering relations with nodes outside its subtree, as shown in Fig 4. *Super pointers* are shortcuts to subscriptions that the node covers. The tree and the super pointers form a directed acyclic graph (DAG) capturing the covering relations among subscriptions. With super pointers, a node covers its subtree, the nodes with subtrees pointed to by its super pointers, and the nodes with subtrees pointed to by its offsprings' super pointers.

The tree is maintained as follows. When a new subscription arrives, a breadth first search traverses the tree in order to find a place to insert the subscription. At a given node the following three cases are distinguished.

Case 1: If the new subscription has no covering relation with the node, the node's siblings are searched. If neither sibling has a covering relation with the new subscription, the subscription is inserted as new sibling, After insertion, super pointers maintained by the parent node are updated. If there is a super pointer of the parent pointing to a subscription that is also covered by the new subscription, then the super pointer is moved from the parent node to the new node.

Case 2: If the new subscription covers the current node, the new subscription is inserted between the current node and its parent. As a result, the new subscription becomes the parent of the current node, the old parent becomes the new subscription's parent. The old parent's super pointers are updated and moved to the new node, if there is a covering relation between the inserted subscription and a subscription pointed to by the super pointer.

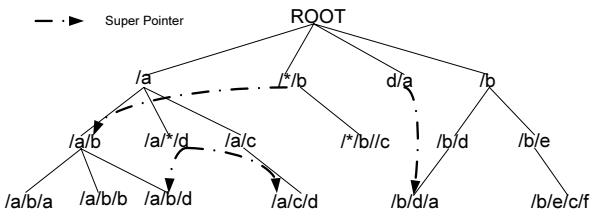


Figure 4. Subscription Tree

Case 3: If the new subscription is covered by the current node, its children are searched until the new subscription is inserted. If the current node is a leaf node, the new node is inserted as the current node's child.

Existing super pointers are maintained while inserting. The new subscription may cause new covering relations and new super pointers are added. Every time a new subscription arrives, we add new super pointers into existing nodes that cover the new subscription while searching the tree. However, this becomes expensive when the subscription tree grows larger. The reason we maintain the updated super pointers is for covering-based routing. When a subscription arrives, if it is not covered by existing subscriptions but it covers a set of subscriptions, we need to unsubscribe the subscriptions it covers and only forward the new subscription to neighbors. In this case, we need the super pointers to tell us what subscriptions should be unsubscribed. That means the updating of super pointers can be postponed to that point. The search space is reduced by super pointers. Note that we only need to unsubscribe subscriptions in the higher level of the tree since nodes in the subtrees are covered and unsubscribed already.

In the worst case, it takes $O(n)$ to identify the covering relations and insert a subscription to the subscription tree. For example, a subscription is covered by all subscriptions which are organized in one path. If the subscription tree created from a subscription workload is balanced, the best case run time is $O(log(n))$, which is the height of the tree.

Optimizations for the subscription searching and insertion can be performed based on the following two properties of the subscription tree.

Property of an Absolute XPE node: For all the absolute simple XPEs which have no wildcard and //-operators, the children's path length is always longer than their parent's path length. The parent is the prefix of its children.

Based on this property, we can perform depth-first search for an XPE to find a start node which has the same length as itself and start breadth-first search at that level. If an absolute XPE has a wildcard or //-operator in the middle of the expression, it is one or more levels higher than other simple XPEs of the same length in the subscription tree. Based on this property we can stop the search earlier.

Property of a Relative XPE node: A relative XPE is a child node of either the root node or another relative node. It will never be inserted in a subtree rooted by an absolute XPE. This property reduces the search space in the subscription tree.

4.2. Covering Algorithm

The key problem is how to determine the relationship between two given subscriptions. The covering relation between subscriptions is the containment problem in the context of XPEs. It has been proven that containment of simple path expressions can be tested in PTIME [23]. It is studied as a part of the problem that checking/finding a prefix replacement for a simple query is in PTIME. In this section, we detect covering relations of XPEs containing wildcard, /- and //-operator in PTIME, and present covering rules and algorithms for determining covering relations between single path XPEs. We say Sub_1 containing an element t_i covers Sub_2 containing an element m_i at the corresponding position, if t_i is a wildcard no matter what m_i is, or $t_i = m_i$, where none of t_i and m_i is a wildcard.

Absolute simple XPEs: The covering relation between two absolute XPEs (without //-operator) is the simplest case. We describe the covering algorithm for two absolute XPEs (`AbsSimCov`) as below (e.g., s_1 and s_2). An important observation is that s_1 must be shorter than s_2 if s_1 covers s_2. This is exploited because a shorter XPE s has less constraints on items in an XML document, and refers to a bigger matching set $P(s)$. Next, the algorithm compares each pair of s_1t_i and s_2t_i in s_1 and s_2, respectively, according to the covering rules.

Relative simple XPEs: The covering algorithm for relative simple XPEs (`RelSimCov`), e.g., s_1 is relative, and s_2 is absolute or relative, calls `AbsSimCov` repeatedly to determine if s_1 contains subscription s_2 or not. An absolute XPE s_1 can not cover a relative XPE s_2, as the absolute XPE definitely refers to a smaller matching set $P(s_1)$ than $P(s_2)$.

It is important to note that the covering algorithm `RelSimCov` is also a string matching problem, as we pointed out in the `RelExprAndAdv` algorithm. The covering algorithm uses covering rules that are different from subscription and advertisement matching rules used in `RelExprAndAdv`, however, a similar optimization can be applied to reduce the complexity of the covering algorithm from $O(k * n)$ to $O(k)$.

Descendant operators in XPEs: In this section, we describe the covering algorithm for XPEs with descendant operators (`DesCov`), where both s_1 and s_2 can be relative or absolute. It splits the XPE into sub-XPEs without //-operator, and matches each sub-XPE in s_1 against sub-XPEs in s_2 with sequence comparisons. First, it guarantees that s_2 is longer than s_1. Next, it matches the first sub-XPE in s_1 against s_2 according to different types of s_1 and s_2. The algorithm moves to the next sub-XPE in s_1 if it finds a match, and moves to the next sub-XPE in s_2 if it does not find a match. For example, given $s_1 = /*/a//*/c$ and $s_2 = /a/a/*//c/e/c/d$, first, the algorithm compares the sub-XPE $/*/a$ in s_1 with the sub-XPE $/a/a/*$ in s_2. Second, it moves to the next sub-XPE $*/c$ in s_1 and compares it with $*$ in s_2. Next, it compares $*/c$ in s_1 with the next sub-XPE $c/e/c/d$ in s_2, and finally, it returns *true* since the end of s_1 is reached and a match is found. Generally speaking, a sub-XPE in s_1 could not match a part of s_2 that includes a //-operator. For instance, given $s_1 = /*/a//*/c$ and $s_2 = /a/a/*//c/b/d$, the sub-XPE $*/c$ in s_1 does not cover $*//c$ in s_2 since $*/c$ refers to a smaller matching set. However, there is a special case that the sub-XPE s_{1i} in s_1 could cover a part of s_2 that includes a //-operator if s_{1i} ended with a wildcard and the matched part in s_2 ended with $//t$, where t can be either a wildcard or an element. For example, given $s_1 = /a/*//*/d$ and $s_2 = /a//b/c/d$, first, the algorithm compares $/a/*$ in s_1 with $/a//b$ in s_2, where $flag = 1$ is used to record the current sub-XPE in s_1 matches a part of s_2 with //-operator. Second, it moves to the next sub-XPE $*/d$ in s_1 and compares it with c/d in s_2. Finally, it returns *true* since a match is found.

It is important to note that the covering detection between non-recursive advertisements is the same with the covering detection for subscriptions, since the non-recursive advertisement has the same format with an absolute simple subscription.

4.3. Merging

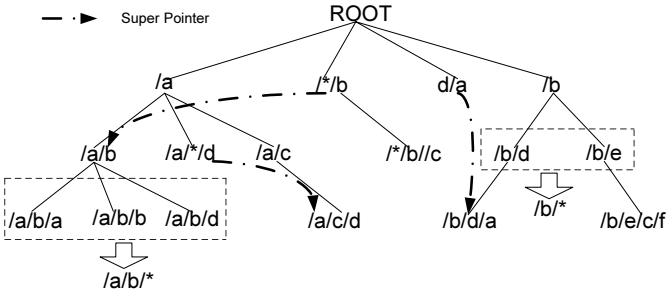


Figure 5. Subscription Tree

If there is no covering relation among a set of subscriptions, subscriptions can be merged into a new subscription to create a more concise routing table. In this section, we exploit the merging rules for XPEs.

In the subscription tree, child nodes of the same parent have a better chance to be merged. As shown in Fig 5, node $/a/b/a$, $/a/b/b$ and $/a/b/d$ can be merged and they are represented by a new node $/a/b/*$ which is a union of the original XPEs. There was a super pointer pointing to node $/a/b/d$ before merging. This pointer should be removed because there is no covering relation between the pointer owner and the merger. If two nodes are merged, their subtrees become siblings of the merger. For example in Fig 5,

after $/b/d$ and $/b/e$ are merged to $/b/*$, their children are the new node's children. The super pointer at $/b/d/a$ was not changed. To perform the merging in the subscription tree, we define several merging rules.

The subscriptions can be merged if they have only one difference (e.g., different elements). For instance, two subscriptions $s_1 = a/*/c/d$ and $s_2 = a/*/c/e$ can be merged into $s = a/*/c/*$. Note that if they differ in one operator, they should be in a covering relation to each other. The general form of this rule is:

- $s_1 = o_1\ t_1 ...\ o_i\ t_i\ o_{i+1}\ m\ o_{i+2}\ t_{i+1} ... o_{n+1}\ t_n$
- $s_2 = o_1\ t_1 ...\ o_i\ t_i\ o_{i+1}\ k\ o_{i+2}\ t_{i+1} ... o_{n+1}\ t_n$

are merged to

- $s = o_1\ t_1 ...\ o_i\ t_i\ o_{i+1}\ *\ o_{i+2}\ t_{i+1} ... o_{n+1}\ t_n$

where m and k are different elements, o_i is either a /-operator or a //-operator, and t_i is a wildcard or an element. The number of merging candidates in this rule is not limited to 2.

Another rule is to merge subscriptions with two differences (e.g., different operators or different elements). For example, two subscriptions $s_1 = /a/c/*/*$ and $s_2 = /a//c/*/c$ that do not cover each other can be merged to $s = /a//c/*/*$. That is, different operators are merged to //-operator, and different elements are merged to $*$. We represent the general form of this rule as:

- $s_1 = o_1\ t_1 ...\ o_i\ t_i\ o_{i+1}\ m ...\ o_{j+1}\ t_j\ /\ t_{j+1} ... o_{n+2}\ t_n$
- $s_2 = o_1\ t_1 ...\ o_i\ t_i\ o_{i+1}\ k ...\ o_{j+1}\ t_j\ //\ t_{j+1} ... o_{n+2}\ t_n$

are merged to

- $s = o_1\ t_1 ...\ o_i\ t_i\ o_{i+1}\ * ...\ o_{j+1}\ t_j\ //\ t_{j+1} ... o_{n+2}\ t_n$

where m, k and t_i is a wildcard or an element, and o_i is a /-operator or a //-operator.

A more general rule is to replace the different parts in two subscriptions with the //-operator. We generalize this rule to:

- $s_1 = o_1\ t_1 ...\ o_i\ t_i\ XPE_1\ o_{i+1}\ t_{i+1} ... o_n\ t_n$
- $s_2 = o_1\ t_1 ...\ o_i\ t_i\ XPE_2\ o_{i+1}\ t_{i+1} ... o_n\ t_n$

are merged to

- $s = o_1\ t_1 ...\ o_i\ t_i\ //\ t_{i+1} ... o_n\ t_n$

where XPE_1 and XPE_2 are different XPath expressions, t_i is a wildcard or an element, and o_i is a /-operator or a //-operator. This rule is applied if most parts in two subscriptions are equal, otherwise, more false positives will be introduced.

We periodically apply the above merging rules on the subscription tree to aggregate nodes that could be merged. We can compute an *imperfect merging degree* if each broker in the network knows the DTD relative to the XML data producer. An imperfect merger was first introduced in [20]. The imperfect degree of a new merger s, derived from s_1, s_2,..., s_n, is:

$$D_{imperfect} = \frac{|P(s) - \cup_{i=1}^{n} P(s_i)|}{|P(s)|}$$

It measures the imperfectness of an individual new merger. If the publications are distributed uniformly, the bigger the imperfect degree, the more false positive are introduced by the new merger. For example, two subscriptions $s_1 = /a/*/c/d$ and $s_2 = /a/*/c/e$ can be merged into $s = /a/*/c/*$. If the corresponding DTD indicates that the elements a, b, c, d, e are allowed at the fourth position, 60% false positive will be introduced at position 4. We need to consider the distribution of other elements in the subscription, e.g., the probability of each element appearing at other positions, to compute the total number of false positive introduced. Based on the DTD information, if $D_{imperfect}$ is 0, the merger is a perfect merger, and no false positives are introduced in this case. The false positives are not delivered to subscribers. They only occur in the network introduced due to imperfect merging. Clients are not exposed to false positives.

5. Evaluation

In this section, we experimentally evaluate the performance of our routing and covering algorithms. All algorithms are implemented in C++. We perform all experiments on a local cluster of 20 nodes and on PlanetLab. Each node in the cluster has an Intel Xeon 2.4GHz processor with 2GB RAM. For generating the XPE workload, we use the XPath generator released by Diao *et al.* [10]. Queries are distinct, and we set the maximum length of an XPE to 10. We use the IBM XML Generator [2] to create the XML document workload. We use default parameters in this generator except that we set the maximum number of levels of the resulting XML documents to 10, which is consistent with the maximum length of XPEs. We use two different DTDs: the NITF (News Industry Text Format) DTD and the PSD (Protein Sequence Database) DTD. The performance metrics we measure include routing table size, XPE processing time and publication routing time in a single broker. We compare the network traffic (i.e., number of messages) and notification delay (i.e., the time between issuing a publication and receiving a notification) in two broker topologies with 7 brokers and 127 brokers, respectively. We also deployed our system on PlanetLab and measured the notification delay to validate the scalability of our approach.

Routing Table Size (RTS): Our algorithms exploit the covering relations among XPEs. To verify this fact, we generate two data sets for NITF which include 100,000 XPEs each. We vary the probability of "$*$" occurring at a location step (W) and the probability of "//" occurring at a location step (DO) to generate two data sets A and B with different covering rates 90% and 50%, respectively. For each data set, we evaluate the effect of the covering optimization on routing table size. The routing table size is the number of XPEs in the table. As shown in Figure 6, for Set A, the rout-

ing table size is reduced dramatically by covering. The subscriptions in Set A have a higher degree of overlap. The results suggest that the covering algorithm performs better on data sets with higher degree of overlap. That is, the covering technique achieves more benefit when subscribers have similar interests.

Merging can further reduce the routing table size by merging some XPEs according to our merging rules. When $D_{imperfect}$ is 0, the merger is a perfect merger. Figure 7 shows that applying perfect merging reduces the routing table size to 87%. When $D_{imperfect}$ increases, more XPEs can be merged. For instance, the routing table is compacted to 67% with $D_{imperfect}$ equal to 0.1.

XPE Processing Time: We measure the XPE processing time of the covering algorithm. In covering-based routing, we first check the covering relationship when an XPE arrives at a broker. If the XPE is covered by existing XPEs, it will not be forwarded. Otherwise, we match the XPE against all advertisements and determine where to route it to. Without covering algorithm, every XPE needs to be matched against all advertisements in order to be forwarded. We issue 5000 XPEs, and Fig. 8 shows the processing time per XPE. Each data point in Fig. 8 is the average processing time for 500 XPEs. Although detecting covering relations takes extra time, the experiment shows that the XPE processing time is less in covering-based routing, which avoids matching the covered XPEs against advertisements. For example, among the 5000 PSD XPEs, 90% of the XPEs are covered. The more XPEs are covered, the greater the improvement we achieve. Covering-based routing improves the XPE processing time of NITF XPEs by up to 49.2%, which is more than for the PSD XPEs. The reason is because the number of advertisements generated from the NITF DTD is 35 times larger than that of the PSD DTD. As a result, we benefit more from avoiding advertisement matching, especially when the broker has a large number of advertisements.

Publication Routing Time: In this experiment, we evaluate the covering-based routing time of each message using data sets A and B. Note that the performance of non-covering-based routing in the original system has been evaluated against YFilter [10] in our previous work [16]. For some scenarios (i.e., the XPE workload with a high percentage of matched expressions, and with many wildcards and descendant operators), our system outperformed YFilter. For a contrasting workload with a very low matching percentage, YFilter outperformed us. We generate 500 XML documents and extract 23,098 publications from these documents. Table 1 shows the routing time of the publications against 100,000 XPEs. The measurements are obtained by averaging the time taken to route all publications. Both Set A and Set B exhibit benefits, derived from subscription covering. After applying the covering algorithm, the routing time for Set A and Set B are reduced by 84.6% and 47.5%, respectively. The merging technique generates a more compact routing table, with which we can further improve the publication routing time.

Method	Set A (ms)	Set B (ms)
No Covering	13.96	14.23
Covering	2.15	7.47
Perfect Merging	1.87	6.88
Imperfect Merging	1.27	6.38

Table 1. Publication Routing Performance

Network Traffic: The network traffic can be influenced by the broker topology, the distribution of subscribers and publishers, and the routing strategy. In this experiment, we investigate the impact of advertisement-based routing and covering techniques on network traffic, given a tree-like broker topology. The broker overlay network is a tree in which each broker is connected to 2 subordinate brokers. We build two overlays for the experiment. One has three levels, which consists of 7 brokers. The other broker overlay has seven levels with 64 leaf brokers, and 127 brokers in total. Each leaf broker is connected with a subscriber. We extend the size of the broker network to show the scalability of our approach. Publishers randomly connect to the broker overlay.

Method	Network Traffic	Delay (ms)
no-Adv-no-Cov	58,138	29.02
no-Adv-with-Cov	50,931	7.50
with-Adv-no-Cov	39,849	28.9
with-Adv-with-Cov	38,492	7.45
with-Adv-with-CovPM	25,789	5.15
with-Adv-with-CovIPM	26,146	3.92

Table 2. 7 Broker Network

Method	Network Traffic	Delay (ms)
no-Adv-no-Cov	654,871	97.82
no-Adv-with-Cov	572,890	20.74
with-Adv-no-Cov	398,810	98.09
with-Adv-with-Cov	326,796	20.89
with-Adv-with-CovPM	254,900	16.78
with-Adv-with-CovIPM	257,567	12.24

Table 3. 127 Broker Network

In this experiment, we compare routing strategies with different optimization techniques, including the routing with neither advertisement nor covering technique (no-Adv-no-Cov), the routing with covering only (no-Adv-with-Cov), the routing with advertisement only (with-Adv-no-Cov), the routing with both advertisement and covering techniques

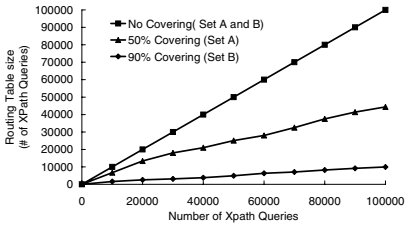


Figure 6. RTS

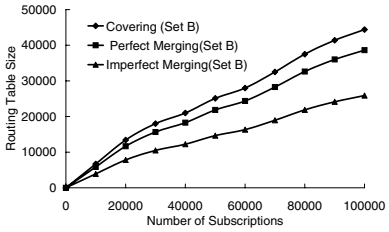


Figure 7. RTS

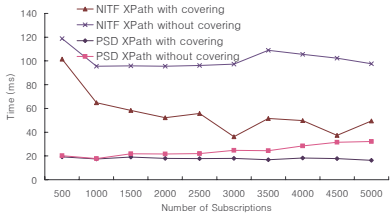


Figure 8. Process Time

(with-Adv-with-Cov), the routing with advertisement, covering and perfect merging (with-Adv-with-CovPM), and the routing with advertisement, covering and imperfect merging (with-Adv-with-CovIPM). We generate 1,000 distinct XPEs for each subscriber using the PSD DTD, and 50 XML documents for the publishers. 4,182 publications are extracted from these documents. Table 2 and Table 3 show the total number of messages in the two broker overlays generated by one publisher. These messages, including advertisements, publications and subscriptions, are received by all brokers in the network under different routing strategies. As can be seen, the two advertisement-based routing methods significantly reduce the network traffic, because in this case a subscription is not flooded, and it is only forwarded to brokers that are on a path from the subscriber to potential publishers. The introduction of advertisements reduces the network traffic to 68.5% and 75.6% for non-covering-based and covering-based routing strategies, respectively. Moreover, applying both advertisement-based routing and covering-based routing techniques can reduce the overall network traffic to 66.2% and 49.9% in the two topologies. The experiment suggests that using advertisements to avoid subscription flooding and removing redundant queries by exploring covering and merging relations among subscriptions can reduce network traffic, save system resources and reduce the publication routing delay. Note that since imperfect merging may introduce false positives, the network traffic due to imperfect merging increases by 1.38% and by 1.04% in the two overlays, respectively. Overall, we achieve more benefit in a larger broker network. The scalability of the system is improved.

False Positives from Imperfect Merging: If the system allows a larger error tolerance, more subscriptions can be merged. An imperfect merger may match more publications than expected. Therefore, a larger $D_{imperfect}$ means that the system allows more false positives. We show the relation between $D_{imperfect}$ and false positives in Figure 9. The larger $D_{imperfect}$ is, the greater the number of matched publications, among which some are false positives introduced by imperfect mergers. If the system tolerates up to 2% of false positives, a $D_{imperfect}$ with value less than 0.1 can satisfy the requirement. False positives only occur in the network and are not delivered to clients. Thus, they induce overhead but do not violate the subscription semantic expected by clients.

Notification Delay on PlanetLab: In this experiment, we measure the notification delay on PlanetLab and demonstrate the scalability of our system. Due to the performance variation on PlanetLab nodes, which in our experiment is up to 15% per data point, we average the results from four experimental runs, as shown in Fig. 10 and Fig. 11. We setup a broker network with the maximum end-to-end distance equal to 7 hops. We measure the notification delay from publishers to subscribers for different number of broker hops and different XML document sizes. The experiment shows that the notification delay in covering-based routing is reduced by up to 74% compared with the routing without covering for both NITF and PSD documents. Moreover, the notification delay is linear in the number of hops. In covering-based routing, it increases less with the number of hops than in the content-based routing without covering. The reason is that the routing table size along the routing path has been reduced by the covering technique, as a result, the XML document matching time at each hop decreases, for instance, the routing table size is reduced to 6% for PSD XPEs. The result also indicates that the larger the document the longer the notification delay. A larger document saves more matching time with a condensed routing table. Therefore, the larger the document, the greater the improvement in routing delay we can achieve from the covering technique.

6. Related Work

A large body of work has focused on developing publish/subscribe-style matching algorithms for evaluating an XML message against a set of XPEs [3, 10, 7, 4]. However, all these approaches exclusively address centralized matching architectures, not the distributed, content-based XML dissemination networks we address in this work. While matching is an integral step in a content-based router, other routing operations studied in this paper are equally

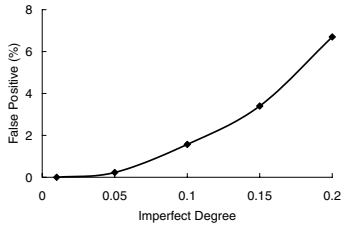


Figure 9. False Positives

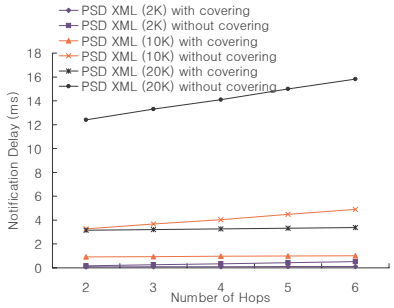


Figure 10. PSD XML

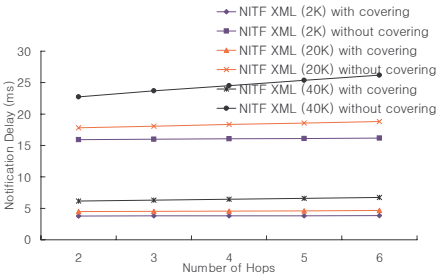


Figure 11. NITF XML

important in a distributed data dissemination architecture. Thus, our work complements matching algorithms for the design of a content-based XML router.

Advertisement-based techniques for optimizing content-based routing have been developed in the area of distributed publish/subscribe [5, 24, 9]. It has been demonstrated that the network traffic and routing table size can be reduced by using different routing strategies, including advertising, covering and merging techniques. However, the main differences between these approaches and our approach lie in the subscription language and publication data format. Our approach is based on the hierarchical, tree-based XPath and XML model; while the traditional content-based routing approaches operate with attribute-value pairs and predicate constraints over these pairs. The advertising carried out by XML and XPath data sources is different and more complex than predicate-based languages, for the hierarchical and recursive structure of the model needs to be taken into account. A DTD of an XML document does not have an equivalent in traditional publish/subscribe approaches. Galanis *et al.* [15] explore XML data dissemination based on a DHT. They use data summaries to ensure that queries are only sent to relevant servers. These data summaries could be taken as a form of advertisements. The data summary is generated from an XML document, so the expressiveness of the data summary is part of the DTD. Our contribution is to generate a complete advertisement set once from a DTD for all related XML documents.

The query aggregation scheme given in [6] addresses the problem of determining a compact set of XPEs from a given set of XPEs. This problem is similar to the covering and merging problem discussed in this paper. We think it could be used for the same purposes. However, it should guarantee the equivalence between the compact set and the original set. That is, the aggregate query set does not introduce or control false positives (i.e., takes XML documents not originally matched) or false negatives (i.e., misses XML documents originally matched.) These are non-trivial extensions to their work. Furthermore, the tree aggregation approach does not address the generation of advertisements from DTDs, which is central to our approach.

Recent research has focused on XML data dissemination [18, 11, 25, 19, 13]. Koloniari *el al.* [18] present a decentralized approach for XML dissemination in a peer-to-peer network. However, in their approach queries are severely restricted in that no wildcards are allowed. Koudas *et al.* [19] propose a flexible routing protocol for XML routers to enable scalable XPath query and update processing in a data-sharing peer-to-peer network. Both approaches are solutions to the *location problem*. The location problem states that given a dynamic collection of XML database servers and an XPath query, find the databases that contain data relevant to the query. Our approach evaluates an XML document against a set of XPath queries, and decides where to route the XML document.

ONYX [11] describes an architecture to deploy XML-based services on an Internet-scale. The approach is based on performing incremental message transformations to reduce message size instead of sending and receiving the entire XML message, published by a data source. ONYX only delivers the parts of the message actually selected by the data sinks' subscribing queries. For many applications, this approach is not feasible, as the entire message published by a source needs to be delivered in its entirety to all subscribing data sinks, which is the message delivery semantic realized by our system and algorithms. To further reduce message size and processing cost, ONYX investigates various representations for XML messages. While a binary message representation will certainly speed up XML message processing, we have found in prior work that XML processing, such as parsing, is not the dominating cost for an XML router [16] and optimizations of that component are therefore of questionable utility in this context. ONYX uses an NFA-based operator network for representing routing tables. This approach supports the sharing of common prefixes among queries. Our approach goes beyond this by identifying all covering relations among queries processed when constructing the routing tables, so that all covered queries are eliminated completely from the routing computation. Moreover, our work introduces merging and adver-

tising for XML data not addressed in the earlier approach. Our unique contribution is to enable these techniques for the XML data model, which is fundamentally different from the data models underlying non-XML-based content routing approaches, such as [5, 24, 9]. XTreeNet [13] elegantly unifies the publish/subscribe and the query/response model in an XML-aware overlay network. XTreeNet proposes a dissemination protocol to avoid repeatedly matching XML data against queries at intermediate routers. This is an orthogonal optimization that our approach can also employ by attaching the path of overlay hops to the XML query when the subscription tree is constructed in the network. XML messages only match against the queries at the first router, and are forwarded along the subscription paths to the subscribers.

Theoretical properties of XPE containment are discussed in [22, 12]. They propose their own algorithms to detect the covering relations and give the computational complexity analysis for these algorithms, but none of them apply advertisement-based routing for XML dissemination. They do not consider XPE merging either.

7. Conclusions

In this paper, we studied the problem of efficiently routing XML data through a data dissemination network comprised of an overlay network of content-based routers. In the dissemination network, publishers' DTD files are transformed into advertisements expressed using XPath-like expressions. An advertisement creates a spanning tree rooted at the publisher. Subscribers specify XPath filters which are forwarded along the reverse paths of this tree for *intersecting* advertisements. XML documents from publishers are forwarded along these routing paths to subscribers with *matching* XPEs. By defining and exploiting covering and merging relations for XPEs, a compact routing table results. Our techniques improve the routing time at each broker by up to 85% in the most favorable cases. Our experiments demonstrate that the scalability of the system is improved by applying advertisement-based routing, covering, and merging techniques for routing XML documents in a data dissemination network.

Acknowledgments: The completion of this research was made possible thanks to Bell Canada's support through its Bell University Laboratories R&D program. This research was also funded in part by CA, CFI, IBM, NSERC, OCE, OIT, and Sun.

References

[1] The PADRES content-based publish/subscribe system web site. `http://padres.msrg.toronto.edu/Padres/`.

[2] A.L.Diaz and D.Lovell. XML generator, Sept. 2003.

[3] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB*, 2000.

[4] N. Bruno, L. Gravano, and N. Doudas. Navigation-vs. index-based XML multi-query processing. In *ICDE*, 2003.

[5] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *INFOCOM*, 2004.

[6] C. Chan, W. Fan, and P. Felber. Tree pattern aggregation for scalable XML data dissemination. In *VLDB*, 2002.

[7] C. Chan, P. Felber, and M. Garofalakis. Efficient filtering of XML documents with XPath expressions. In *ICDE*, 2002.

[8] A. K. Y. Cheung and H.-A. Jacobsen. Dynamic load balancing in distributed content-based publish/subscribe. In *Middleware*, 2006.

[9] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructrue and its application to the development of the OPSS WFMS. *IEEE TSE*, 2001.

[10] Y. Diao, M. Altinel, and M. J. Franklin. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans. Database Syst.*, 2003.

[11] Y. Diao, S. Rizvi, and M. Franklin. Towards an internet-scale XML dissemination service. In *VLDB*, 2004.

[12] X. Dong, A. Halevy, and I. Tatarinov. Containment of nested XML queries. In *Technical Report UW-CSE-03-12-05, Univ. of Washington*, 2003.

[13] W. Fenner, M. Rabinovich, K.K.Ramakrishnan, D. Srivastava, and Y. Zhang. XTreeNet: Scalable overlay networks for XML content dissemination and querying (synopsis). In *Proceedings of the 10th International Workshop on Web Content Catching and Distribution*, 2005.

[14] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. The PADRES distributed publish/subscribe system. In *ICFI*, 2005.

[15] L. Galanis, Y. Wang, S. Je, and E. DeWitt. Locating data sources in large distributed systems. In *VLDB*, 2003.

[16] S. Hou and H.-A. Jacobsen. Predicate-based filtering of XPath expressions. In *ICDE*, 2006.

[17] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 1977.

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

[19] N. Koudas, M. Rabinovich, and D. Srivastava. Routing XML queries. In *ICDE*, 2004.

[20] G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *ICDCS*, 2005.

[21] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Middleware*, 2005.

[22] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 2004.

[23] T. Milo and D. Suciu. Index structures for path expressions. In *ICDT*, 1999.

[24] G. Mühl. Large-scale content-based publish/subscribe systems. *Ph.D Dissertation*, University of Darmstadt, September 2002.

[25] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-based content routing using XML. *SIGOPS Oper. Syst. Rev.*, 2001.