Efficient XML Data Dissemination with Piggybacking

Chee-Yong Chan
National University of Singapore
chancy@comp.nus.edu.sg

Yuan Ni
National University of Singapore
niyuan@comp.nus.edu.sg

ABSTRACT

Content-based dissemination of XML data using the publish-subscribe paradigm is an effective means to deliver relevant data to interested data consumers. To meet the performance challenges of content-based filtering and routing, two key optimizations have been developed: the use of efficient indexes to speed up subscription filtering, and the use of effective aggregation algorithms to reduce the number of subscriptions. The effectiveness of both these techniques are, however, limited to locally improving the performance of individual routers. In this paper, we propose a novel and holistic optimization approach that allows a downstream router to leverage the subscription matchings done by upstream routers to reduce its own filtering work. This is achieved by piggybacking useful annotations to the XML document being forwarded. We explore several design options and tradeoffs of this novel optimization approach. Our experimental results demonstrate that our piggyback optimization achieves significant performance improvement under various conditions.

Categories and Subject Descriptors: H.2.4 [Information Systems]: Systems - Query processing

General Terms: Algorithms, Design, Performance

Keywords: annotation, data dissemination, piggybacking, pub/sub system, XML, XPath

1. INTRODUCTION

The ubiquity of XML data and the effectiveness of the content-based pub/sub-based paradigm [23] of delivering relevant information has led to a lot of interest in content-based dissemination of XML data (e.g., [8, 12, 11]). In the pub/sub environment, an overlay network of application-level *routers* (or message brokers) is used to asynchronously forward documents generated by *data publishers* to relevant *data subscribers* (or consumers) based on matching the document contents against the consumers' subscriptions. Each subscriber first needs to pre-register a *subscription* to its local router, specifying the type of data that he is interested in receiving. A routing protocol (e.g., [13]) is used to set up a *routing table* at each router to record both its local subscriptions as well as the subscriptions from its neighboring routers. For each document received by a router, the router compares the document contents against the subscriptions in its routing table and forwards the document to the matching local subscribers and neighboring routers.

Content-based data dissemination has traditionally been examined in the context of simple subscription specifications, such as matching of keywords or conjunction of basic comparison predicates on attribute values (e.g., [7, 24, 10]). The recent interest in using the more expressive XPath-based subscriptions for effective dissemination of XML documents has increased the complexity of the content-based subscription matching problem. Thus, there is a even greater need for effective optimization techniques to meet the performance challenges of content-based dissemination of XML data. The existing research efforts have focused on two key optimizations to minimize the number of subscription matchings. The first optimization, which has attracted the most attention, is to exploit efficient index structures (e.g., [12, 14, 8, 9, 16, 17, 19, 21, 22, 26]) to perform selective matching with only a small subset of potentially matching subscriptions. The second optimization uses aggregation algorithms to summarize an initial set of subscriptions into a smaller set of generalized subscriptions (based on subscription containment properties) to reduce the number of subscriptions and the matching overhead [11, 26].

The effectiveness of the existing optimizations are, however, limited to only locally improving the performance of the individual routers. Specifically, the fact that routers are interconnected and related (in terms of the containment relationships of their subscriptions) is not being fully exploited to optimize the subscription matching.

To appreciate the new optimization opportunities and motivate our approach, consider how a document D is being routed from an upstream router R_i to a downstream router R_j in a typical pub/sub system. On receiving D, R_i parses and processes D against the set of subscriptions S_i stored in its routing table. Once a matching subscription $s \in S_i$ (that is associated with R_j) is detected, R_i then forwards D to R_j. A similar processing of D is then repeated at R_j but with the matching now being done against a different set of subscriptions S_j in R_j's routing table. Our new optimization idea is motivated by the following two observations on the matching and routing process. Firstly, the overall processing being done at the different routers during the dissemination of a document can be viewed as essentially processing the same data (i.e., XML document) against a sequence of collections of queries (i.e., sets of subscriptions along each path of forwarding routers). Secondly, the sequence of collections of queries being processed are not independent as they are partially related by a "containment property" that determines whether or not a document is to be forwarded to a downstream router. Specifically, the sets of subscriptions S_i and S_j are related in that the subscriptions S_j in the downstream router are being aggregated (or summarized) into a smaller set of subscriptions S'_j that is stored in the upstream router R_i's routing table (i.e., $S'_j \subseteq S_i$) such that if a document D does not match any of the subscriptions in S'_j, then D will certainly not match any of the subscriptions in S_j (i.e., S'_j is "contained by" by S_j). Consequently, R_i needs to forward D to R_j only if D matches some subscription in S'_j.

Thus, given that the same document D is being processed against related sets of subscriptions, each upstream router R_i can help to optimize the performance of its downstream router R_j (and thereby reduce the overall processing time to deliver D to relevant subscribers) by passing along some useful information to R_j (about D as well as the about related queries that R_i has processed) when it forwards D to R_j. R_j can then try to exploit the hints that it receives from R_i to optimize its own processing of D. There are two key ways that a downstream router R_j can optimize its processing and matching of D by exploiting additional hint information from its upstream router R_i:

1. The hint could enable R_j to quickly determine that D be forwarded to a downstream router R_k without requiring R_j to parse and process D.

2. The hint could enable R_j to quickly detect that a portion $S'_j \subseteq S_j$ of the subscriptions in R_j's routing table are guaranteed not to match D, and R_j can therefore speed up its matching of D against the smaller set $(S_j - S'_j)$ instead of S_j.

We illustrate the above optimization opportunities with the following two examples.

Example 1.1 Consider the following routing of a document D among three routers (R_i, R_j, and R_k), where D is first forwarded from R_i to R_j due to a matching subscription $/a//d \in S_i$ (that is associated with R_j); and then D is then forwarded from R_j to R_k due to a matching subscription $/a/b/c/d \in S_j$ (that is associated with R_k). Observe that if R_i had forwarded to R_j (along with D) the additional information on the data bindings for the matching subscription $/a//d \in S_i$; i.e., that the wildcard "//" in $/a//d$ actually matches the data path "b/c" in D, then R_j could have very efficiently determined that D matches the subscription $/a/b/c/d \in S_j$ without actually having to parse and process D against the subscriptions in S_j. In this way, R_j is able to speed up the forwarding of D to R_k and thereby reduce the overall processing time to disseminate D to relevant subscribers. □

Example 1.2 Consider the scenario where a router R_i needs to forward a document D to its downstream router R_j, and that after having parsed and processed D against S_i, R_i has obtained the following information about D and S_i: (H1) D does not match some subscription $s \in S_i$ that is associated with R_j; (H2) the data pattern "x/y/z" occurs in D with its last occurrence located at some position p within D; and (H3) the data pattern "a/b/c" does not occur at all in D. Observe that each of these three pieces of information could be forwarded to R_j as hints to optimize the performance of R_j. For (H1), R_j can use the non-matching subscription $s \in S_i$ to identify the subset of subscriptions $S'_j \subseteq S_j$ in R_j that were aggregated to s (i.e., subscriptions that are guaranteed to not match D), and exclude matching D against such subscriptions to improve the matching performance. For (H2), once R_j has parsed D beyond position p, R_j can conclude that there will not be any new matches of subscriptions that contain the data pattern $x/y/z$, and therefore such subscriptions can be excluded from further matching and processing. Finally, (H3) can be treated as a special case of (H2) with p being at the starting position of the document D. Thus, R_j can ignore matching D against the subscriptions in S_j that contain the data pattern $a/b/c/$ right at the beginning of D. □

In this paper, we propose an orthogonal and holistic optimization that enables a downstream router to leverage the subscription matching work completed by upstream routers to optimize its own performance. To facilitate such "collaborative" processing, a router is allowed to piggyback some additional useful information (that is derived from its subscription processing) as header annotations to an XML document before forwarding it to a downstream router. On receiving an annotated document, a router first pre-processes the header annotations to check if the hints specified in the annotations could enable any immediate forwarding decisions (without processing the document) or if they could be used to reduce the effective number of subscriptions in its router table that need to be matched against the document. We refer to this approach to optimize performance as *piggyback optimization*.

Note that this new optimization could be used in combination with the existing optimizations proposed for pub/sub systems (i.e., subscription indexing and aggregation).

There are three key design issues to be addressed for our piggyback optimization approach:

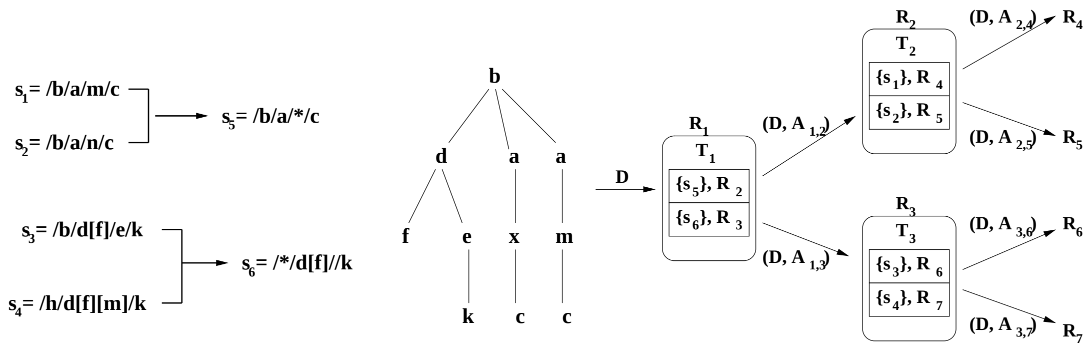
1. What type of information is useful to piggyback?

2. How can such information be efficiently computed by a forwarding router and exploited by a receiving router?

3. How does this optimization impact the data matching protocol (i.e., when a router R_i detects that some subscription corresponding to a downstream router R_j matches a document D, should R_i forward D immediately to R_j? And should R_i continue matching D against other subscriptions related to R_j?)

As there are many possible types of hints that could be forwarded along with a document, forwarding too much hints could increase both the transmission cost as well as the overhead of pre-processing the hints and thereby possibly negating the potential performance improvements. Thus, the hints need to be selected judiciously to balance these tradeoffs. Intuitively, a hint is preferred if it is more likely to be beneficial and can be efficiently computed by the upstream router and exploited by the downstream router.

In this paper, we examine and evaluate several design options for piggyback optimization. Our experimental study demonstrates that our proposed piggyback optimization is



(a) Aggregating and aggregated subscriptions (b) XML document, D (c) Routing tables

Figure 1: XPath Subscriptions, XML Document, and Routing Tables

indeed a feasible and effective technique to improve the performance of content-based dissemination of XML data. In particular, our results show that the overhead of creating and preprocessing annotations is low relative to the performance improvement obtained from minimizing the number of subscription matchings using the annotated information. Our results indicate that the piggyback optimization is effective to improve the performance at various conditions, especially when the matching occurs at the late part of the document, the performance is improved by a factor of 2.

Contributions. The key contributions of this paper include: (1) the proposal of a novel, holistic optimization technique for XML data dissemination called *piggyback optimization* that enables upstream routers to pass useful hints (in the form of document header annotations) to optimize the performance of downstream routers. This new optimization is orthogonal to the existing indexing and subscription aggregation optimizations; (2) a thorough study of the design options and issues for piggyback optimization; and (3) a comprehensive experimental performance evaluation demonstrating the efficiency of piggyback optimization.

Organization. The rest of this paper is organized as follows. Section 2 presents further background and introduces terminologies and notations. Section 3 presents our piggyback optimization technique. We discuss the mechanism to generate the annotations in Section 4 and the approach to process the annotated document in Section 5. Section 6 presents experimental results, and related work is discussed in Section 7. We conclude our paper in Section 8.

2. BACKGROUND AND NOTATIONS

In this section, we provide some further background on content-based XML dissemination and introduce some related terminologies and notations.

Pub/Sub systems. In a pub/sub environment, each data consumer registers his subscription to his local router. In order for a router to know about subscriptions that have been registered with other routers, a routing protocol is used by the routers in the overlay network to exchange subscription information so that their subscription tables are set up correctly to establish routing paths for forwarding documents.

We use R_i to denote a router, and T_i to denote the set of subscription entries in its routing table (refer to Fig. 1(c)). Conceptually, each entry in T_i is of the form (S_j, p_j), where S_j denotes a set of subscriptions and p_j denotes a unique identifier that refers to either a local subscriber of R_i or a neighboring router of R_i. For a given document D, we use $S_j^+(D)$ and $S_j^-(D)$ to denote, respectively, the subset of subscriptions in S_j that matched and did not match D (i.e., $S_j = S_j^+(D) \cup S_j^-(D)$). For each incoming document D to R_i, R_i will forward D to p_j if and only if $S_j^+(D)$ is non-empty.

If a router R_i forwards some document to a neighboring router R_j, we call R_i an *upstream router* and R_j a *downstream router*. In order for any document to be forwarded from an upstream router R_i to a *downstream router* R_j, R_j needs to have advertised (via some routing policy) its collection of subscriptions (i.e., $U_j = \bigcup_{(S,p)\in T_j} S$) to R_i so that an entry (U_j, R_j) can be recorded in T_i.

XPath subscriptions. Subscriptions on XML data are typically specified using the expressive XPath-based patterns [25]. In this paper, we focus on a commonly used and expressive fragment of XPath that uses only the child ("/") and descendant ("//") axes. The node test for each XPath step can be either an element name or a wildcard "*". Nested XPath expressions are also allowed as predicates. Fig. 1(a) gives some sample XPath expressions.

Subscription aggregation. Since the entire collection of subscriptions in R_i (i.e., $U_i = \bigcup_{(S,p)\in T_i} S$) is generally large, R_i needs to summarize (or aggregate) U_i to a smaller set $\mathcal{S}'_i$ of aggregated subscriptions before advertising it to its neighboring routers. To preserve forwarding correctness, $\mathcal{S}'_i$ needs to satisfy the following *containment property* w.r.t. U_i: for every document D, if D matches some subscription $s \in U_i$, then there must exist some subscription $s' \in \mathcal{S}'_i$ such that D also matches s'. We say that $\mathcal{S}'_i$ *contains* U_i (or U_i is *contained by* $\mathcal{S}'_i$), denoted by $U_i \sqsubseteq \mathcal{S}'_i$. Similarly, we say that a subscription s' contains another subscription s, denoted by $s \sqsubseteq s'$, if $\{s\} \sqsubseteq \{s'\}$. The importance of the containment property (i.e., $U_i \sqsubseteq \mathcal{S}'_i$) is that using $\mathcal{S}'_i$ in place of U_i for document matching will guarantee that there are no *false negatives* (i.e., documents not being forwarded when they should); however, *false positives* can arise (i.e., documents being forwarded when they need not) which are tolerable and do not compromise correctness.

Several algorithms (e.g., [11, 26]) have been developed to aggregate a set of subscriptions S into a smaller set S' such that $S \sqsubseteq S'$, and they are all formulated (at a high level) in terms of the following two steps: first, partition S into a collection of disjoint subsets $S_1, \cdots, S_m$, where $m < |S|$; next, aggregate each S_i into a single subscription s'_i (i.e., $S_i \sqsubseteq \{s'_i\}$) to obtain $S' = \{s'_1, \cdots, s'_m\}$ with the properties that $S \sqsubseteq S'$ and $|S'| < |S|$. In addition, to ensure that the aggregated subscriptions are space-efficient, a space bound is generally impose on S' to limit the total number of query steps among all the queries in S'.

For each of the subscriptions $s \in S_i$, $S_i \sqsubseteq S'$, that becomes aggregated to $s'_i \in S'$ (i.e., $s \sqsubseteq s'_i$), we refer to s as an *aggregating subscription*, and refer to s'_i as an *aggregated subscription* of s. A subscription s' is said to be a *simple aggregated subscription* of another subscription s (or s is a *simple aggregating subscription* of s') if (1) $s \sqsubseteq s'$, and (2) s' can be made to match s by simply substituting each wildcard (i.e., * and //) in s' with some path of data element names.

Example 2.1 Consider the set of XPath expressions $S = \{s_1, s_2, s_3, s_4\}$ in Fig. 1(a). One way to aggregate S into a smaller set is to first partition S into two subsets $S_1 = \{s_1, s_2\}$ and $S_2 = \{s_3, s_4\}$; followed by aggregating S_1 and S_2, respectively, into s_5 and s_6 as shown in Fig. 1(a). It can be verified that $S_1 \sqsubseteq \{s_5\}$ and $S_2 \sqsubseteq \{s_6\}$. We say that s_5 and s_6 are, respectively, the aggregated subscriptions of S_1 and S_2; and the subscriptions in S_1 and S_2 are aggregating subscriptions. Note that s_6 is a simple aggregated subscription of s_3 (by substituting the * and // in s_6, respectively, with elements b and e), but s_6 is not a simple aggregated subscription of s_4 (since no substitution in s_6 can create the path d/m in s_4). □

3. PIGGYBACK OPTIMIZATION

In this section, we present a novel approach to optimize the subscription matchings at a router. Our technique, which we termed *piggyback optimization*, is orthogonal to the two existing optimizations, namely, indexing techniques and subscription aggregation algorithms, that are also targeted at improving the routers' performance.

As motivated in the introduction, the central idea behind piggyback optimization is to optimize the performance of a router by leveraging information from the work done by its upstream router. This is possible because both the upstream and downstream routers are processing the same document against subscriptions that are partially related (due to subscription containment relationships). Thus, an upstream router could pass to its downstream router some useful hints (along with the document being forwarded) about properties of the document and/or matching/non-matching subscriptions that it has encountered to enable the downstream router to optimize its performance by expediting the forwarding of the document (without processing the document) and/or speeding up its subscription matching process.

In our proposed piggyback optimization, the hints from an upstream router are disseminated to its downstream router in the form of header annotations in the document. On receiving an annotated document, a router will first preprocess the header annotations to optimize the subsequent processing of the document. There are three key design issues that need to be addressed for our piggyback optimization approach: (1) What type of information is useful to piggyback? (2) How can such information be efficiently computed by an upstream router and exploited by its downstream router? (3) How does this optimization impact the matching protocol?

	Subscription	**Data**
Positive	Data bindings for matching subscriptions	Positions of last occurrences of data patterns
Negative	Non-matching subscriptions	Non-occurring data patterns

Figure 2: Types of Annotations

In this paper, we use $A_{i,j}$ to denote the header annotations that an upstream router R_i adds to a document D before forwarding it to a downstream router R_j. The annotated document that R_j receives from R_i is denoted by (D, $A_{i,j}$).

3.1 Types of annotations

The first key issue for the piggyback optimization technique is to decide on what types of information to include in the header annotation of a document to optimize performance. Our design of the annotated information is guided by three performance-related requirements. Firstly, it should be concise so that it incurs minimal processing overhead in terms of parsing and transmitting the additional header information. Secondly, it should be efficiently generated so that the computation overhead incurred by the upstream router does not offset any performance gains of its downstream routers. Thirdly, it should be effective in that a downstream router can efficiently preprocess the annotations to optimize its subscription matching performance.

Let us consider the possible sources of useful information that an upstream router R_i can pass on to a downstream router R_j along with a document D that needs to be forwarded to R_j.

After having matched its own subscriptions against D, R_i has acquired additional information about D and how its subscriptions are related to D. We can classify this knowledge into *positive* and *negative* information:

- *Positive information* refers to information about (a) subscriptions in T_i that matched D, and (b) patterns / properties that occur in D.
- *Negative information* refers to information about (a) subscriptions in T_i that did not match D, and (b) patterns/properties that did not occur in D.

We shall refer to the annotations of these two types of information as *positive annotations* and *negative annotations*.

In the following, we identify two types of positive annotations (PS and PD) and two types of negative annotations (NS and ND), which are classified in Figure 2. The emphasis of the discussion in this section is on the ideas; we address the implementation issues in Section 4.

3.1.1 *Positive annotations*

A positive annotation specifies information related to either (1) a matching subscription or (2) a data pattern that occurs in the document. Subscription-related information could be used to expedite a document forwarding decision without having to process the document itself, while data-related information could be used to reduce the effective number of subscriptions that need to be matched.

Positive subscription (PS). A PS annotation is of the form (s_x, B_x), where s_x is some subscription detected by R_i to match D (i.e., $s_x \in S_j$, $(S_j, R_j) \in T_i$), such that s_x is a simple aggregated subscription of some subscription in T_j; and B_x is the set of pairs (l, p) such that l specifies the position of a wildcard (i.e., * and //) in s_x and the p is the binding of that wildcard corresponding to the detected matching. Thus, a PS annotation essentially specifies a data pattern in D that matches some subscription in T_i. Such an annotation can benefit R_j if the specified data pattern also matches some subscription s_y in T_j that aggregates to s_x. When this happens, R_j can very quickly detect that D matches s_y (from processing $A_{i,j}$) without having to actually process D (which is more costly to process than $A_{i,j}$). R_j can then immediately forward D to the relevant downstream router.

Example 3.1 Continuing with Example 2.1, consider the processing of the document D in Fig. 1(b) by the router R_1, which has two immediate downstream routers R_2 and R_3. R_1 first detects that D matches the subscription s_6 with the wildcard * and // in s_6 matching data elements b and e in D, respectively. R_1 then adds the PS annotation $(s_6, \{(1, b), (2, e)\})$ into $A_{1,3}$ and forwards the annotated document $(\mathrm{D}, A_{1,3})$ to R_3. Next, R_1 detects that D also matches the subscription s_5 with the * in s_5 matching the data element x. R_1 then adds the PS annotation $(s_5, \{(1, x)\})$ into $A_{1,2}$ and forwards $(\mathrm{D}, A_{1,2})$ to R_2. On receiving $(\mathrm{D}, A_{1,3})$, R_3 will first check whether any of the aggregating subscriptions for s_6 (i.e., s_3 and s_4) matches s_6 with the bindings $\{(1, b), (2, e)\}$. In this case, there is indeed a matching for the subscription s_3. Thus, R_3 can very quickly forward D to the router R_6 without having to scan and process D. On the hand, when R_2 receives $(\mathrm{D}, A_{1,2})$, none of the subscriptions in R_2 (i.e., s_1 and s_2) matches the PS $(s_5, \{(1, x)\})$ in $(\mathrm{D}, A_{1,2})$. In this case, R_2 needs to scan and process D before detecting that s_1 matches D. Note that if R_1 had created a PS for the second matching of s_5 in D as well, R_2 would have been able to detect the matching of s_1 earlier without having to process D. □

Positive data (PD). The purpose of a PD annotation is to specify some useful property about the data D that can potentially be exploited by a downstream router R_j to skip the matching of some of its subscriptions in T_j thereby reducing R_j's processing overhead.

In this paper, we use a simple PD of the form (p, l), where $p = e_1/e_2/\cdots/e_m$ refers to a path of element names that exists in D, and l refers to the position of the last occurrence of p in D. Here, the position information l means that the end-tag of the element e_1 in the last occurrence of p is l^{th} end-tag in D. To see how a PD (p, l) can be exploited, suppose R_i has just completed parsing the subtree of data elements rooted at the l^{th} element in D, then R_i can safely ignore all of the subscriptions that contain the pattern p from further processing since such subscriptions are guaranteed not to match the remaining yet-to-be-processed portion of D. This subscription pruning optimization can improve performance particularly if the location l is early or if there are many subscriptions in T_i that contain p.

3.1.2 Negative annotations

The main idea behind negative annotations is to identify the set of subscriptions in the downstream router that are guaranteed not to match the document D being forwarded. In this way, the downstream router can optimize its performance by eliminating the need to compare against such subscriptions against D.

Negative subscription (NS). The NS annotation for a downstream router R_j (w.r.t. D) is a list of the identities of all the non-matching subscriptions in S_j (i.e., $S_j^-(D)$). This information can be exploited by R_j to skip the matching of all the aggregating subscriptions that were aggregated to $S_j^-(D)$. Specifically, for each subscription s in T_j, if s is an aggregating subscription of an aggregated subscription $s' \in S_j^-(D)$ (i.e., $s \sqsubseteq s'$), then by the containment property, the fact that D did not match s' at R_i necessarily implies that D will not match s at R_j. Thus, the matching of s against D at R_j is redundant and can be skipped without affecting correctness.

Negative data (ND). Besides using non-matching aggregated subscriptions to skip the matching of corresponding aggregating subscriptions, a more general approach to enable a similar optimization is to exploit the absence of certain data patterns in D to skip the matching of all subscriptions in R_j that contain such patterns. As an example, if R_i knows that D does not contain the path of elements $p = A/B$, this negative information can be beneficial to R_j if there is a large collection C_p of subscriptions in T_j that contain such a pattern p. By similar reasoning using the containment property, R_j can safely skip the matching of the subscriptions in C_p. In this paper, we use simple data patterns in the form of linear paths of element names for ND annotations.

3.2 Impact on Matching Protocol

The *matching protocol* of a router R_i refers to the two key decisions that R_i makes when it detects that some subscription corresponding to a downstream router R_j matches D. The first deals with whether R_i should forward D immediately to R_j; and the second deals with whether R_i should continue matching D against other subscriptions related to R_j.

In this section, we discuss the impact of piggyback optimization on the options for the matching protocol.

3.2.1 Eager forwarding with skipping (ES)

For routers in conventional pub/sub systems (without piggyback optimization), the matching protocol adopted is that when a document D is detected at an upstream router R_i to match some subscription associated with some downstream router R_j, R_i will immediately forward D to R_j and skips the matching of subscriptions associated with R_j. Forwarding a document as soon as possible helps to improve response time, while skipping unnecessary subscription matchings helps to reduce the processing overhead. We refer to this conventional protocol as *eager forwarding with skipping* (denoted by ES).

3.2.2 Lazy forwarding without skipping (L)

The conventional ES approach of forwarding D to a downstream router R_j as soon as a subscription matching for R_j is detected generally occurs when D has not been completely processed. The eager forwarding protocol has two implications with regards to the use of annotations. First, negative annotations cannot be included in the forwarded document; and second, only limited PS annotations (derived from the processed portion of D) can be used. Given that negative annotations could potentially help to skip a large number of subscriptions in R_j and PS annotations could enable a document to be forwarded quickly without processing the document, it might actually be beneficial to delay the forwarding of D to R_j until D has been completely processed at R_i. Clearly, for this "lazy" forwarding protocol to generate additional annotations for a matching downstream router R_j, it is necessary for R_i to continue matching D against the subscriptions that correspond to R_j. We refer to this protocol as *lazy forwarding without skipping* (denoted by L).

There is a performance tradeoff between the ES and L protocols. On the one hand, by forwarding D immediately to a downstream router, ES can help to reduce the response time of delivering a document to matching data consumers. On the other hand, by delaying the forwarding to generate both negative as well as complete PS annotations, L can potentially minimize the matching cost at each downstream router by (1) using negative annotations to skip the processing of many subscriptions, and (2) using PS annotations to enable D to be quickly forwarded without having to first parse D. L is particularly cost-effective if at the time a subscription matching is detected at R_i, only a small proportion of D has not been processed.

3.3 Combining annotations and protocols

Based on the preceding discussion, the design space of our piggyback optimization consists of four basic annotation types (PS, PD, NS, and ND) and two matching protocols (ES and L). A data dissemination strategy is formed by choosing a subset of annotation types together with a matching protocol. We use P^{α}_{β} to denote a dissemination strategy, where $P \in \{ES, L\}$ refers to the matching protocol used; $\alpha \subseteq \{+s, +d\}$ refers to the set of positive annotations used; and $\beta \subseteq \{-s, -d\}$ refers to the set of negative annotations used. Here, $+s$, $+d$, $-s$, and $-d$ denote, respectively, PS, PD, NS, and ND annotations. For conciseness, we use $+sd$ (resp., $-sd$) to represent $\{+s, +d\}$ (resp., $\{-s, -d\}$); moreover, an empty set value is simply represented by a blank, and a singleton value $\{x\}$ is abbreviated to x. For example, the conventional dissemination strategy is denoted by ES; and a strategy that uses lazy forwarding with PS, PD, and ND annotations is denoted by L^{+sd}_{-d}.

Note that it is not meaningful to have a dissemination strategy that involves some negative annotation type together with the ES policy. This is because negative annotations cannot be added to D if it is eagerly forwarded when the processing of D has not completed. Therefore, in this paper, we do not consider dissemination strategies ES^{α}_{β} with $\beta \neq \emptyset$.

4. GENERATING ANNOTATIONS

In this section, we discuss the details of how an upstream router R_i computes the various annotations (i.e., $A_{i,j}$) for a data D to be forwarded to a downstream router R_j. Except for NS annotations, all the other annotations in $A_{i,j}$ can be created more effectively if they exploit knowledge of the subscriptions in the downstream router R_j. To achieve this, our approach generates PS, PD, and ND annotations in two steps, referred to as the *offline step* and *online step*. The offline step is performed only once as part of the routing protocol to set up the routing tables in the routers. Specifically, when a downstream router R_j is advertising its aggregated subscriptions to each of its upstream routers, we also make use of this opportunity to transmit some useful information that is derived from R_j's subscriptions to the upstream routers. This derived information from R_j will be stored by the upstream routers and used to create annotations in the online step for documents that are forwarded to R_j. The online step is performed by an upstream router each time it needs to forward a document to some downstream router.

In general, since there are many possible options for each annotation type, we devise a benefit metric for each annotation type to enable the effectiveness of annotations to be compared so that a reasonably small set of beneficial annotations can be judiciously selected (for inclusion in the document header) that both maximizes the performance improvement for the downstream router as well as ensures transmission efficiency.

In the following subsections,we describe, for each of the four annotation types, the benefit metric used to select annotations and how the annotations are created.

4.1 Positive subscription (PS)

Intuitively, a PS annotation (s, B) is beneficial for a downstream router R_j if s is a simple aggregated subscription of many subscriptions in R_j as this increases the chance that the pattern specified by (s, B) matches a subscription in R_j. On the other hand, it is also desirable for the size of the binding B to be small so that the annotation is space-efficient. We therefore define the benefit of using a subscription s for a PS annotation in $A_{i,j}$ to be $benefit(s) = \frac{|S'|}{\sum_{i \in w(s)} size(i)}$ where S' is the set of simple aggregating subscriptions of s in R_j, $w(s)$ is the set of the wildcards in s, and $size(i)$ is size of binding values for the i^{th} wildcard of s. Note that a subscription s has no benefit if $S' = \emptyset$. R_i will select the most beneficial PS annotations based on the above metric.

PS annotations to be included in $A_{i,j}$ are computed in two steps. In the offline step, R_j will identify a set of candidate subscriptions that can be used for PS annotations and advertise these candidates to R_i. In the online step, whenever R_i needs to forward a document to R_j, R_i will create PS annotations from a subset of these candidates by adding relevant data bindings.

More specifically, in the offline step, after R_j has aggregated its subscriptions, R_j derives the information (s, $|S'|$, l) for each simple aggregated subscription s computed by R_j, where S' is the subset of subscriptions in R_j that was aggregated to s such that s is a simple aggregated subscription of each subscription in S' (defined in Section 2); and l is a list of wildcard positions indicating which of the wildcards (// and *) in s need to be instantiated with data path bindings to match some aggregating subscription in S'.

Example 4.1 Consider the example in Fig. 3, where the 6 subscriptions in R_j are partitioned into three sets: $S_1 = \{s_1, s_2\}$, $S_2 = \{s_3, s_4\}$, and $S_3 = \{s_5, s_6\}$, which are aggregated, respectively, into s'_1, s'_2, and s'_3. And for simple aggregated subscriptions s'_1 and s'_3, $S'_1 = \{s_1, s_2\}$ and $S'_3 = \{s_5\}$. The derived information generated by R_j in the offline step for these aggregations are shown as i_1, i_2, and i_3 in Fig. 3. Observe that $i_1 = (s'_1, |S'_1|, l_1)$, where $|S'_1| = 2$ and $l_1 = \{2\}$ indicating that only the second wildcard (i.e., *) requires a data binding; however, the first wildcard (i.e., //) does not require a data binding to transform s to any of the subscriptions in S. On the other hand, since s'_3 is the simple

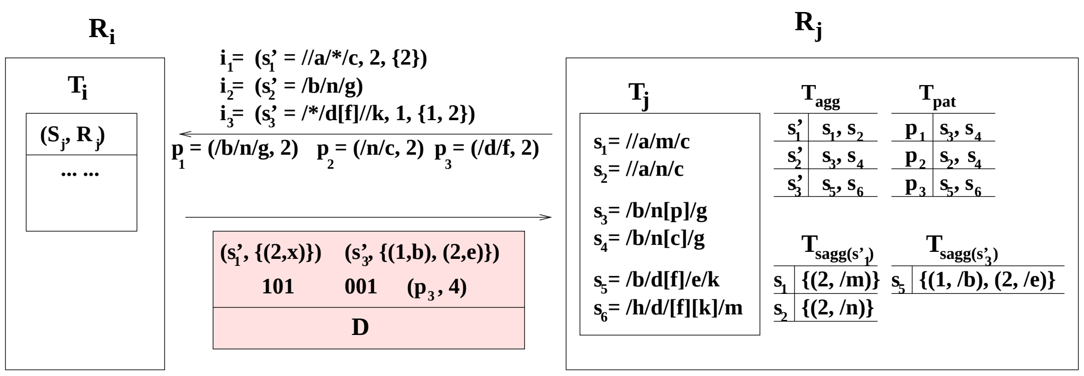


Figure 3: Generating & Processing Annotations

aggregated subscription of only $s_5 \in S'_3$, $i_3 = (s'_3, |S'_3|, l_3)$, where $|S'_3| = 1$ and $l_3 = \{1, 2\}$. □

The collection of derived information $(s, |S'|, l)$ will be passed to upstream routers of R_j when R_j advertises its aggregated subscriptions to them. In the online step, when an upstream router R_i detects a matching subscription s (associated with downstream router R_j) while matching a document D, R_i will first compare s against the derived information from R_j to determine whether s could form a candidate PS annotation.

Example 4.2 Continuing with Example 4.1, suppose that R_i in Fig. 3 is processing the document D from Fig. 1(b). When R_i detects that subscription s'_1 matches the data $/b/a/x/c$ in D, R_i uses the derived information $i_1 = (//a/*/c, 2, \{2\})$ to create the PS annotation $(s'_1, \{(2, x)\})$ for s'_1. Based on i_1, R_i knows that only the second wildcard in s'_1 requires a data binding. On further processing D, R_i detects that subscription s'_3 matches D and creates the PS annotation $(s'_3, \{(1, b), (2, e)\})$ for this matching. Fig. 3 shows the scenario where R_i is forwarding all these PS annotations to R_j along with D (indicated by the shaded box). □

4.2 Positive data (PD)

A PD annotation (p, l) is beneficial for R_j if p is contained by many subscriptions in T_j and the value of l is small so that R_j can skip many of its subscriptions after processing only a small portion of D. We therefore define the benefit of a pattern p to be $\frac{freq(p)}{pos(p)}$, where $freq(p)$ represents the number of subscriptions in T_j that contain p; and $pos(p)$ represents the position of the last occurrence of p in D. R_i then selects a subset of the PD annotations of the form $(p, pos(p))$ that have the highest benefit values.

PD annotations are also computed in two steps. In the offline step, R_j advertises to R_i a small set of beneficial data patterns P_j (together with their frequencies) derived from the subscriptions in T_j. R_j selects data patterns to be included in P_j based on the following benefit metric for a data pattern p: $benefit(p) = freq(p) \cdot \ln(l(p) + 1)$, where $freq(p)$ represents the number of subscriptions in T_j that contain p; and $l(p)$ represents the length of the linear pattern p. The function $\ln(l(p)+1)$ provides an approximate measure of the probability that the last appearance of p occurs early in the document. In the online step, an upstream R_i keeps track of $pos(p)$ for each pattern $p \in P_j$ as it processes D. With the $freq(.)$ information from R_j and the $pos(.)$ information that it derives, R_i can approximately select the most beneficial PD annotations for $A_{i,j}$.

Example 4.3 Consider again the example in Fig. 3 which shows that R_j is advertising three candidate data patterns p_1, p_2 and p_3 (along with their frequencies) to R_i for possible use as PD annotations. In the online step, after R_i has completed processing D (from Fig. 1(b)), R_i detects that $pos(p_3) = 4$ (note that there is no occurrence of p_1 and p_2 in D). It decides to create the PD annotation $(p_3, 4)$ and forwards it to R_j. □

4.3 Negative subscription (NS)

An NS annotation is a subscription $s \in T_i$ that did not match D; and it is more beneficial to R_j if there are more subscriptions in T_j that aggregate to s as it enables R_j to skip the processing of a larger number of subscriptions. The benefit of each $s \in T_i$ is therefore defined to be the number of subscriptions in T_j that aggregates to s. NS annotations are computed in two steps. In the offline step, R_j notifies to its upstream router R_i the number of its subscriptions that aggregate to each aggregated subscription. While processing a document D during the online step, R_i selects from among the subscriptions in T_i that did not match D, the subset with the highest benefit values as NS annotations. However, as the total number of subscriptions is generally not too large, all the non-matching subscriptions can be specified concisely and precisely using a bitstring with each subscription represented by a single bit such that the bit is turned on if and only if the subscription is non-matching.

4.4 Negative data (ND)

An ND annotation is of the form of a linear data pattern p that is absent in D. Intuitively, a data pattern p is more beneficial to R_j if there are more subscriptions in T_j that contain p and the probability of p's occurrence in D is low. In fact, ND annotations can be viewed as a special case of PD annotations with $pos(p) = 0$. Thus, we can use the same metric $freq(p) \cdot \ln(l(p) + 1)$ (defined in Section 4.2 for PD annotations) to compare the benefit of different ND annotations. The generation of ND annotations in $A_{i,j}$ follows a similar two-step process, where R_i advertises to R_j a set of candidate data patterns during the offline step; and during the online step, R_i keeps track of the candidate patterns that did not occur in the document being processed, and concisely represent the non-matching data patterns as a bitstring in the ND annotations.

Example 4.4 After R_i in Fig. 3 has processed the document D from Fig. 1(b), R_i detects that D did not match the subscription s'_2 and that the data patterns p_1 and p_2 did not occur in D. Thus, R_i can create the ND annotation 001 and the NS annotation 101 for R_j. □

5. PROCESSING ANNOTATED DOCUMENTS

In this section, we describe the details of how a router R_j processes an annotated document $(\text{D}, A_{i,j})$ that it receives from some upstream router R_i.

To efficiently and effectively process annotations in an annotated document $(\text{D}, A_{i,j})$, each downstream router R_j maintains the following information:

Aggregation Table, T_{agg}. For each aggregated subscription s' generated by R_j, $T_{agg}(s')$ stores the set of aggregating subscriptions of s'.

Simple Aggregation Table, T_{sagg}. For each aggregated subscription s' generated by R_j, if s' is also a simple aggregated subscription, $T_{sagg}(s')$ stores the set of pairs (s, P), where s is a simple aggregating subscription of s'. P is a set of pairs (l_i, p_i), where l_i specifies the position of a wildcard in s' and p_i specifies a data pattern binding such that if for each $(l_i, p_i) \in P$, the l^{th} wildcard in s' is replaced by the pattern p_i, then the transformed s' will match s.

Pattern Table, T_{pat}. For each data pattern p that R_j has advertised to its upstream routers during the offline step for PD or ND annotations, $T_{pat}(p)$ stores the subset of subscriptions in R_j that contains p such that if a document D does not match p, then D also does not match any of the subscriptions in $T_{pat}(p)$.

Non-Matching Array, A_{not}. A_{not} is a bit-array of size equal to the number of subscriptions in R_j such that the i^{th} bit corresponds to the i^{th} subscription in R_j. Each $A_{not}[i]$ is initialized to zero at the start when R_j receives D, which could be set to one as R_j processes the annotations and D. Specifically, $A_{not}[i]$ is set to one if and only if the i^{th} subscription in R_j is guaranteed not to match the document D being processed. R_j uses A_{not} to optimize its processing of D by skipping the processing of subscriptions that indicated by A_{not} to be guaranteed to not match D.

Fig. 3 shows all the tables maintained at R_j for the six subscriptions. Note that all the tables T_{agg}, T_{sagg} and T_{pat} are created only once in the offline step after R_j has advertised his aggregated subscriptions and derived information to its upstream routers. These tables remained static unless there are changes to the subscriptions in R_j. The bit-array A_{not} is the only structure that needs to be initialized and updated for each document that R_j processes.

5.1 Processing Annotations $A_{i,j}$

We are now ready to explain how R_j processes an annotated document $(\text{D}, A_{i,j})$ that it receives from R_i using the following four steps.

Step 1: Processing PS Annotations. For each PS annotation $(s', B) \in A_{i,j}$, R_j compares (s', B) against each $(s, P) \in T_{sagg}(s')$. If for each pair $(l_i, p_i) \in P$, there exists a pair $(l'_i, p'_i) \in B$ such that $l_i = l'_i$ and p_i matches p'_i, then D is detected to match subscription s and R_j can immediately forward D to the downstream router (say R_k) associated with subscription s without the need to first process D[1]. Since R_j has not yet processed D, $A_{j,k}$ needs to be derived from $A_{i,j}$; the details are described in Section 5.3.

Example 5.1 Suppose that R_j has just received from R_i the annotated document $(\text{D}, A_{i,j})$ (indicated by the shaded box in Fig. 3) and is processing the PS annotations. For the PS annotation $PS_1 = (s'_3, \{(1, b), (2, e)\}) \in A_{i,j}$ in which $B = \{(1, b), (2, e)\}$, R_j will process PS_1 against each $(s, P) \in T_{sagg}(s'_3)$. For the tuple $s = s_5$ and $P = \{(1, /b), (2, /e)\} \in T_{sagg}(s'_3)$, R_j detects that it matches PS_1 since for $(1, /b) \in P$ there exists a pair $(1, b) \in B$ such that b matches $/b$; and for $(2, /e) \in P$, there exists the pair $(2, e) \in B$ and e matches $/e$. Thus R_j knows D matches s_5 without processing D. □

Step 2: Processing NS Annotations. For each NS annotation $s' \in A_{i,j}$, R_j knows that D will not match any of the subscriptions in $T_{agg}(s')$. R_j therefore updates $A_{not}[i]$ to one for each aggregating subscription $s_i \in T_{agg}(s')$.

Step 3: Processing ND Annotations. For each ND annotation $p \in A_{i,j}$, R_j knows that D will not match any of the subscriptions in $T_{pat}(p)$. R_j therefore updates $A_{not}[i]$ to one for each subscription $s_i \in T_{pat}(p)$.

Step 4: Processing PD Annotations & D. For each PD annotation $(p, \ell) \in A_{i,j}$, R_j will dynamically process each of them (in ascending order of their positions ℓ) as part of its processing of D. Specifically, if R_j has completed parsing some data element in D at position ℓ and (p, ℓ) is the next-to-be-processed PD annotation, then R_j knows that the data pattern p will not occur in the remaining portion of D, and that it is redundant to process any new matchings of subscriptions in $T_{pat}(p)$. Therefore, R_j updates $A_{not}[i]$ to one for each subscription $s_i \in T_{pat}(p)$.

5.2 Processing Document D

Based on the preceding discussion, R_j processes all the PS, NS, and ND annotations before it begins to process D (steps 1-3). In some situations (step 1), it is even possible for R_j to forward D to another downstream router without having to process D at all. The PD annotations are processed (step 4) along with the normal processing of D.

When R_j detects that D matches some subscription $(S_k, R_k) \in T_j$, there are two cases to consider. If the dissemination strategy used is the ES matching protocol, then R_j will generate the appropriate annotations for $A_{j,k}$ (depending on the annotation types being used) and forward $(\text{D}, A_{j,k})$ to R_k. Moreover, R_j will also skip all subscriptions associated with R_k from further matching as it continues to process D; this is achieved by setting the appropriate bits in A_{not} to one. Otherwise, if the dissemination strategy in use is the L matching protocol, then R_j will only forward D to R_k (with appropriate $A_{j,k}$) after it has completely processed D.

5.3 Deriving Negative Annotations

In this section, we consider the scenario where R_j is going to forward D to a downstream router R_k without having yet completed processing the entire document D. An issue that arises from this situation is what are the possible negative annotations (if any) that R_j can include in $A_{j,k}$ given that R_j has not processed some portion of D. Observe that R_j cannot arbitrarily include a subscription s that has not matched the processed portion of D as an NS annotation

[1]Note that the immediate forwarding due to a matching in $A_{i,j}$ is independent of the forwarding policy being used which applies when there is a matching in D.

since s could potentially have matched D if R_j has processed D completely.

It turns out that R_j can actually derive some limited types of negative annotations for $A_{j,k}$: (1)for each NS annotation $s' \in A_{i,j}$, R_j can identify the subset of subscriptions $S_k \subseteq T_{agg}(s')$ that are associated with R_k. Thus, the subscriptions in S_k can be included as NS annotations in $A_{j,k}$; (2)the ND annotations in $A_{i,j}$ can be directly inherited as ND annotations in $A_{j,k}$. However, since the inherited annotations are not specifically optimized for R_k using derived information from R_k, they are generally less beneficial than the customized ND annotations.

6. PERFORMANCE STUDY

To verify the effectiveness of our proposed annotations in content-based dissemination of XML data, we conducted extensive experiments to compare the performance of various dissemination strategies. Our results show that the dissemination strategy L^{+s}_{-sd} outperforms the conventional method ES by a factor of 2.

6.1 Experimental Testbed

We extended the NS2 network simulator [3] for our experiments by adding application code for content-based routing and piggyback optimization. The subscription indexing method and subscription aggregation approach implemented for each router are based on existing solutions in the literature [12, 11]. It is important to emphasize that our proposed piggyback optimization approach is orthogonal to the specific algorithms used for filtering and aggregation.

The router network topology used was a complete binary tree with fours levels and a total of 15 routers, where each router (except for the root router) has one immediate upstream router at one level above. Data is disseminated from the root router downwards to the leaf routers. Data users can subscribe to any router; and each router aggregates all its subscriptions to a size that is k% of the original set; our experiments used values of 12.5 and 25.0 (the default value) for k. The network bandwidth values used were 1MBps, 10MBps (default), and 100MBps.

Data sets. Our experiments used three synthetic data sets (1) NITF DTD [5], which has been used in previous studies [8, 14, 12]; (2) Treebank [6]; (3) DBLP [2]. For each data set, ten documents were generated using IBM's XML Generator [1]. In addition, we also used one real-life Protein data set [4] by extracting data from it to form small documents. The average sizes of the documents in each of the data sets are shown in Table 1, where NITF$_2$ is the default data set. Note that the document sizes used in our experiments are typical for data dissemination settings [14, 12].

NITF$_1$	NITF$_2$	NITF$_3$	Treebank	DBLP	Protein
50	129.4	200.8	144.1	116.6	276081

Table 1: Average document size used (# elements)

Subscriptions. The subscriptions were generated using the XPath generator in [14], and the parameter values used are shown in Table 2 with the default values used indicated in bold.

In order to study the effect of the choice of forwarding policy on performance, it is important to be able to control the position in the document at which the first subscription matching is detected. Intuitively, the benefit of early forwarding is more significant if the first subscription matching occurs early in the document; while the benefit of lazy forwarding is more significant if the first subscription matching occurs late in the document. To enable the first subscription matching position to be varied, we inserted an unique element <test> in each generated document (varying the location being inserted), and also added an additional predicate $//test$ to each generated subscription. In this way, a subscription matching can occur only after the <test> element has been parsed in the document. In our experimental graphs, the first matching position is represented as a fraction $f \in [0, 1]$ indicating the proportion of the document parsed before the occurrence of a first subscription matching. The values used for f are 0.25, 0.5, 0.75, and 1.

Parameter	Description	Value
L	maximum number of steps	8
ρ_*	probability of "*"	0.1
$\rho_{//}$	probability of "//"	0.1
ρ_λ	probability of nested paths	0.1,**0.2**,0.4
θ	skewness of element names	0,**0.5**
P	#subscriptions per node	2500,**5000**,50000

Table 2: Parameter values for subscriptions

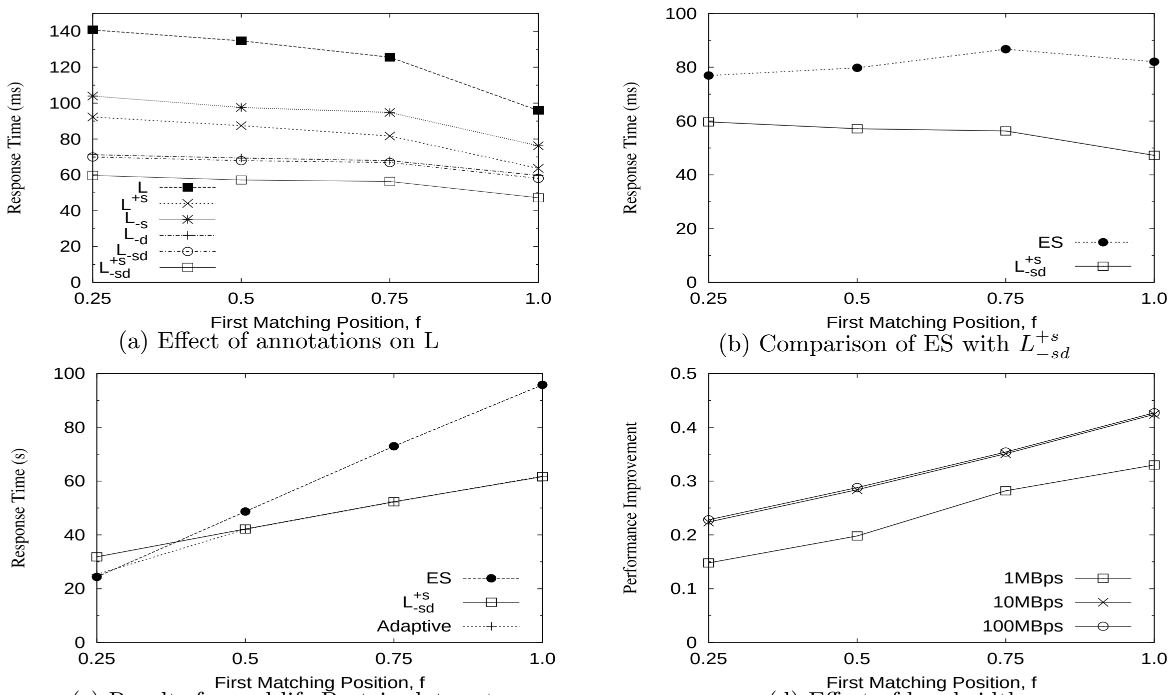
Algorithms. We compared the various dissemination strategies P^{α}_{β} discussed in Section 3.3 by varying the annotation types (α and β) and matching protocols (P), including the conventional approach ES as a special case. The main performance metric compared is the *response time*, which is defined as the average time taken to disseminate a published document from its source to the relevant users. The response time for disseminating a document D comprises of two key components: (1) the transmission spent in in the network, and (2) the processing time incurred by the routers to process annotations against D, match subscriptions against D, and generate annotations. Each response time reported for a data set is the average response time for disseminating the ten documents in the data set.

Our experiments were conducted on a 3GHz Intel Pentium IV machine with 1GB main memory running Windows XP, and all algorithms are implemented in C++.

6.2 Experimental Results

Effect of annotation types. Fig. 4(a) compares the performance of different annotation types using the lazy forwarding without skipping protocol as the first matching position, f, is being varied on the x-axis. Among the four annotation types, ND improves L the most, followed by PS, NS, and PD. L^{+d} (not shown in Fig. 4(a)) turns out to have similar performance as L, since the effectiveness of PD to skip the matching operations is limited. L_{-d} performs better than L_{-s} for the reason that the NITF DTD has many optional elements such that it is likely that some elements or substrings absent from the documents which makes ND more effective to eliminate non-matching subscriptions. Specifically, for our default setting, 10% of the subscriptions on each router are matching. ND can eliminate 70% of the total subscriptions while NS can eliminate only 37% of them. Our results also show that PS is more effective than NS but less effective than ND.

Observe that when both NS and ND are combined, it im-



(a) Effect of annotations on L (b) Comparison of ES with L^{+s}_{-sd} (c) Results for real-life Protein data set (d) Effect of bandwidth

Figure 4: Experimental results for different dissemination approaches

proves over ND only slightly. The reason is that a number of subscriptions that can be skipped using NS can also be skipped by ND; consequently, using NS in addition to ND offers very little improvement. However, as the overhead of using NS is small, adopting both negative annotations is still better than using only a single negative annotation. The performance of L_{-sd} can be further optimized by also using PS, and L^{+s}_{-sd} is in fact the best strategy based on lazy forwarding. This is because PS enables a document to be forwarded to some routers very quickly without parsing the document; and when this is not possible, the negative annotations are effective in skipping many subscription matchings.

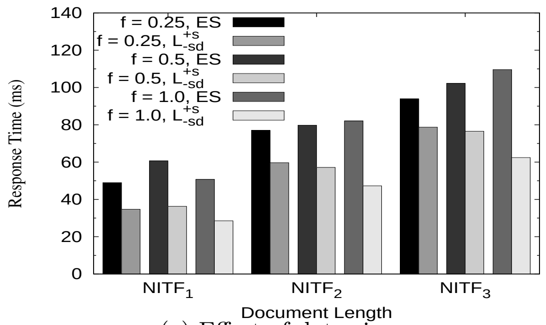
On the other hand, our experimental results (not shown) indicate that positive annotations[2] do not enhance the performance of the eager forwarding policy at all: ES^{+s} has similar performance as ES, while ES^{+d} actually performs worse than ES. This is because only very limited PS annotations can be used when a document is forwarded eagerly; and similar to the case for lazy forwarding, PD turns out to be not cost-effective due to the fact that PD only enables a small number of subscriptions to be skipped and its benefit is offset by its processing overhead.

Eager vs. lazy forwarding. Fig. 4(b) compares the performance of the best eager-forwarding strategy (ES) and the best lazy-forwarding strategy (L^{+s}_{-sd}). The results show that L^{+s}_{-sd} outperforms ES indicating that the slight delay incurred by lazy forwarding is compensated by the improvement gained by the downstream routers from exploiting a more complete set of annotations to optimize their processing. We also observe that as f increases from 0.25 to 1, the improvement by L^{+s}_{-sd} over ES also increases from 22% to 42%. The reason is that as f grows, the lazy forwarding in L^{+s}_{-sd} incurs relatively smaller delay.

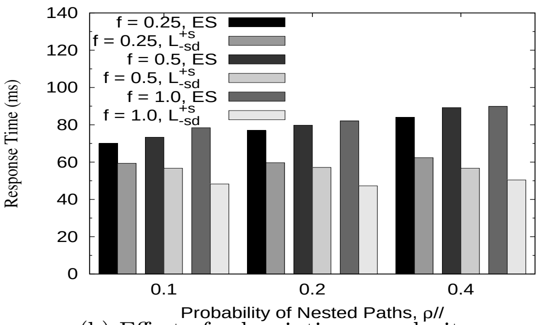
[2]Recall from Section 3.3 that negative annotations are not meaningful for eager forwarding.

Effect of other workload. Our results with other synthetic data sets (Treebank and DBLP) show similar trends with L^{+s}_{-sd} obviously outperforming ES, thus we omit the charts here. Fig. 4(c) shows the comparison using the real-life Protein data set. Observe that ES is actually better than L^{+s}_{-sd} when $f = 0.25$, but as f increases, L^{+s}_{-sd} outperforms ES with increasing margin. The reason that ES performs better with $f = 0.25$ is due to the large document size (about 10MB) : the benefit of the annotations is offset by the longer delay incurred by L^{+s}_{-sd} when f is very small. We also tried an adaptive approach where ES is used when the first subscription matching occurs early and L^{+s}_{-sd} is used otherwise. Our result shows that this hybrid strategy (indicated as "Adaptive" in Fig. 4(c)) outperforms ES and L^{+s}_{-sd}.

Effect of bandwidth. Fig. 4(d) shows the effect of the network bandwidth. With a small bandwidth of 1MBps, L^{+s}_{-sd} has a 15% improvement over ES when $f = 0.25$; which increases to over 30% when f increases to 1.0. As the bandwidth increases to 10MBps, the performance improvement increases to 22% when $f = 0.25$ and to 42% when $f = 1.0$. This is because the disadvantage of L^{+s}_{-sd} (in terms of the delay in forwarding) becomes even less significant (relative to the processing time incurred by the routers) with a higher network bandwidth. Not surprisingly, our results for a bandwidth of 100MBps are similar to the results for a bandwidth of 10MBps. Note that, the space overhead incurred by all annotations used in L^{+s}_{-sd} is 464bytes (the size of PS, NS, and ND are 224, 160, and 80 bytes, respectively) for the default experimental setting where the size of the documents is



(a) Effect of data size



(b) Effect of subscription complexity

Figure 5: Experimental results for varying the document size and query complexity

around 7000bytes. We can see that the size occupied by the annotations is no more than 7% of the document size. Thus, transmitting the additional annotations with the document only incurs a very small overhead in the network delay, while the speedup obtained for the processing on the routers is up to a factor of 2.

Effect of data size. Intuitively, a larger document has two conflicting effects on the performance of L^{+s}_{-sd}. On the one hand, the delay incurred by L^{+s}_{-sd} is expected to increase, but on the other hand, the improvement obtained from skipping subscription matchings would also become more significant with a larger document. Our experimental results shown in Fig. 5(a) (using data sets NITF$_1$, NITF$_2$ and NITF$_3$) indicate that for larger documents (e.g. NITF$_3$) when $f = 0.25$, the improvement of L^{+s}_{-sd} over ES becomes smaller (i.e. 16%), compared with the improvement for NITF$_2$ (i.e. 22%). But their performance margin widens significantly as the value of f increases. When $f = 1$, the improvement of L^{+s}_{-sd} over ES for NITF$_3$ is 41%, which is almost the same with NITF$_2$ (i.e. 42%).

Effect of subscription complexity. By varying the ρ_λ parameter to increase the complexity of the subscriptions, we observe that the performance gain of L^{+s}_{-sd} over ES, shown in Fig. 5(b), becomes more significant with more complex subscriptions. In particular, when ρ_λ increases from 0.1 to 0.4, the improvement of L^{+s}_{-sd} over ES (with $f = 1.0$) increases from 38% to 44%. The reason is because the processing cost of the subscriptions increases with their complexity; thus, the savings from skipping subscription matchings also become more significant.

Effect of number of subscriptions. When the number of subscriptions P is increased from 2500 to 5000, we observe that the improvement of L^{+s}_{-sd} over ES becomes larger (from 35% for P = 2500 to 42% for P = 5000). This is due to an increase in the number of subscriptions that can be skipped by NS and ND. Fig. 6 shows the results when P is increased to 50000. We observe that for larger f (i.e. $f = 0.75$ and $f = 1.0$), the improvement of L^{+s}_{-sd} over ES stays the same (e.g., 41% at $f = 1.0$). However, for smaller values of f, the improvement of L^{+s}_{-sd} over ES diminishes slightly; e.g., with $f = 0.5$, the improvements decreases from 28% for P = 5000 to 11% for P = 50000. When $f = 0.25$, ES is more efficient than L^{+s}_{-sd}. The reason is because for L^{+s}_{-sd}, the root router needs to match against all the subscriptions; and when $f = 0.25$, the delay at the root router is increased

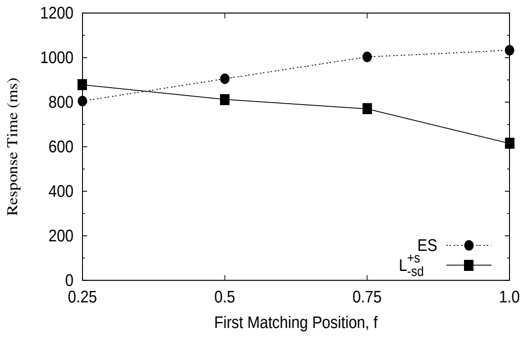


Figure 6: Effect of number of subscriptions

more significantly for L^{+s}_{-sd} (relative to ES) such that the the overall efficiency of L^{+s}_{-sd} becomes diminished.

Other experiments. We also vary other parameters ($k = 12.5$, $\theta = 0$) separately with the following results :(1) With a smaller value of k, PS becomes slightly less effective since the more brute aggregation makes the simple aggregation requirements harder to satisfy; and the performance of NS decreases a little as well due to the less accuracy of the aggregation; (2) Our results show that varying the distribution to generate the subscriptions has little effect on the performance trend.

Throughput comparison. Although L^{+s}_{-sd} performs better in terms of the response time, the results show that the throughput of L^{+s}_{-sd} is slightly worse than ES : when $f = 0.25$, the throughput of L^{+s}_{-sd} is 72% of that of ES; when $f = 1.0$, it increases to 88% of ES's throughput. The total throughput is determined by the slowest router in the system. In L^{+s}_{-sd}, the root router is the bottleneck since there are no annotations that can be used and all subscriptions have to be evaluated, thus the throughput of L^{+s}_{-sd} loses to ES. However, at all downstream routers, the disseminated documents are processed much faster using L^{+s}_{-sd} than using ES because of the effective annotation inserted. Therefore, if we use a more powerful router at the root node, L^{+s}_{-sd} can also achieve higher throughput than ES.

6.2.1 Discussions

Based on our experimental results, we have the following observations on the effectiveness of the various annotation types. First, the effectiveness of PD annotations is found to be limited as only very few subscriptions can be pruned using PD annotations and the marginal saving from using PD annotations is offset by its overhead. Second, comparing ND and NS annotations, the former is generally more effective than the latter. This is due to a combination of two reasons. As the disseminated documents are small and the data schema has many optional elements, many of the elements in the schema (and hence also appearing in many queries) do not occur in the small documents. Thus, ND annotations can help prune many queries in a downstream router. Moreover, the process of aggregating subscriptions often results in a document D matching an aggregated subscription s' in an upstream router even when all of the aggregating subscriptions of s' in a downstream router do not match D. Finally, the relative effectiveness between ND and PS annotations is less clear due to the different nature of their benefits. Recall that the benefit of ND annotations is in reducing subscription matching in a downstream router, while the benefit of PS annotations lies in enabling the forwarding of document without processing it. Neither ND nor PS annotations are found to be significantly more effective than the other in our experiments.

7. RELATED WORK

Work on optimizing the performance of content-based dissemination of XML data has mainly focused on minimizing the number of subscription matchings via two techniques: indexing methods to enable selective subscription matchings [8, 14, 17, 19, 26, 16, 22, 9, 21], and subscription aggregation techniques to summarize a set of subscriptions into a smaller set of generalized subscriptions [11, 26]. A recent interesting direction explored by Fisher and Kossmann [15] examines batched processing of multiple documents which is distinct from our work.

The recent work by Gupta, Halevy, and Suciu [18, 20] also deals with using document header annotations in content-based data dissemination; however, the problem definition and setting are completely different. In their work, each router has a workload of simple conjunctive queries that need to be processed on each incoming document. Their motivation is to try to avoid parsing the document at each router by using a set of views whose results are pre-computed and stored as the document header annotation. Each view's result is represented by a byte offset in the XML document; and the header annotation for a document is statically determined by a centralized server that has knowledge of all the routers' workloads. Our paper is fundamentally different from theirs in the following four ways: (1) our work is motivated by using annotations to leverage the processing done by upstream routers to optimize subscription matchings at downstream routers; (2) our annotations are not static byte offsets but refer to data patterns and subscriptions which can be changed as the document is being routed; (3) our annotations are not precomputed by a centralized entity, but are instead computed dynamically by each forwarding router; and (4) our annotations exploit the containment property between aggregated and aggregating subscriptions for optimization.

8. CONCLUSIONS

In this paper, we have proposed a novel approach to optimize the performance of content-based dissemination of XML data by piggybacking useful annotations to the document being forwarded so that a downstream router can leverage the processing done by its upstream router to reduce its own processing overhead. We have examined a large design space of dissemination strategies that combine four types of annotations and two forwarding policies. Our experimental study demonstrates both the feasibility and effectiveness of the new approach. In particular, the strategy of combining lazy forwarding with three types of annotations turn out to be the best option that outperforms the conventional method by a factor of 2. As part of future work, we intend to explore the performance tradeoffs of adopting other annotation types.

9. REFERENCES

[1] A. L. Diaz, D. Lovell (1999) XML Generator. http://www.alphaworks.ibm.com/tech/xmlgenerator.

[2] DBLP. http://www.acm.org/sigmod/dblp/db/about/dblp.dtd.

[3] NS2. http://www.isi.edu/nsnam/ns/.

[4] Protein. http://pir.georgetown.edu.

[5] R. Cover (1999) The SGML/XML web page. http://www.oasis.open.org/cover/sgml-xml.html.

[6] Treebank. http://www.cis.upenn.edu/ treebank/.

[7] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC*, 1999.

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

[9] N. Bruno, L. Gravano, N. Koudas, and D.Srivastava. Navigation- vs. index-based XML multi-query processing. In *ICDE*, 2003.

[10] A. Carzaniga, D. Rosenblum, and A. Wolf. Design and evaluation of a wide-area event notification service. *ACM TOCS*, 19(3), 2001.

[11] C.-Y. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi. Tree pattern aggregation for scalable XML data dissemination. In *VLDB*, 2002.

[12] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. *VLDB Journal*, 11(4), 2002.

[13] R. Chand and P. A. Felber. A scalable protocol for content-based routing in overlay networks. In *NCA*, 2003.

[14] Y. Diao, M. Altinel, M. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM TODS*, 28(4), 2003.

[15] P. M. Fischer and D. Kossmann. Batched processing for information filters. In *ICDE*, 2005.

[16] X. Gong, W. Qian, Y. Yan, and A. Zhou. Bloom filter-based XML packets filtering for millions of path queries. In *ICDE*, 2005.

[17] T. J. Green, G. Miklau, M. Onizuka, and D.Suciu. Processing XML streams with deterministic automata. In *ICDT*, 2003.

[18] A. Gupta, A. Halevy, and D. Suciu. View selection for XML stream processing. In *WebDB*, 2002.

[19] A. Gupta and D. Suciu. Stream processing of XPath queries with predicates. In *SIGMOD*, 2003.

[20] A. Gupta, D. Suciu, and A. Halevy. The view selection problem for XML content based routing. In *PODS*, 2003.

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

[22] J. Kwon, P. Rao, B. Moon, and S. Lee. Fist: Scalable XML document filtering by sequencing twig patterns. In *VLDB*, 2005.

[23] D. S. Rosenblum and A. L. Wolf. A design framework for internet-scale event observation and notification. In *ESEC/FSE-5*, 1997.

[24] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with Elvin4. In *AUUG2K*, 2000.

[25] W3C. XML path language (XPath). 1999. http://www.w3.org/TR/xpath.

[26] X. Zhang, L. H. Yang, M. L. Lee, and W. Hsu. Scaling SDI systems via query clustring and aggregation. In *DASFAA*, 2004.