Congestion Avoidance with Incremental Filter Aggregation in Content-Based Routing Networks

Mingwen Chen[1], Songlin Hu[1], Vinod Muthusamy[2], Hans-Arno Jacobsen[3]
[1]Chinese Academy of Sciences, [2]IBM T.J. Watson Research Center, [3]University of Toronto

***Abstract*—The subscription covering optimization, whereby a general subscription quenches the forwarding of more specific ones, is a common technique to reduce network traffic and routing state in content-based routing networks. Such optimizations, however, leave the system vulnerable to unsubscriptions that trigger the immediate forwarding of all the subscriptions they had previously quenched. These subscription bursts can severely congest the network, and destabilize the system. This paper presents techniques to retain much of the benefits of subscription covering while avoiding bursty subscription traffic. Heuristics are used to estimate the similarity among subscriptions, and a distributed algorithm determines the portions of a subscription propagation tree that should be preserved. Evaluations show that these mechanisms avoid subscription bursts while maintaining relatively compact routing tables.**

I. Introduction

Route summarization is a common technique used by routers to reduce routing table size and improve routing performance. In IP-networks, addresses are allocated in a hierarchical manner, and it is natural for routers to aggregate a set of addresses based on this hierarchy. For example, all nodes in the $10.0.0.0/8$ subnet can be represented by a single routing table entry for the entire subnet [1].

This summarization is also popular in distributed content-based publish/subscribe systems. For example, *subscription covering* is a common filter summarization technique in content-based publish/subscribe systems, which exploits similarity among subscriptions and takes advantage of the property that a router need not index a specific subscription when there is a more general one present [2], [3]. Consider a content-based router with a general subscription S and a set of more specific subscriptions that are covered by S. The covering optimization will only forward S and avoid forwarding the covered subscriptions. In this way, the covering optimization has many benefits including smaller routing tables, fewer subscription propagations, and faster routing decisions [3], [4].

Unlike IP routers, content-based routers cannot rely on a static hierarchical subnet structure. Instead, node "addresses", that is, their content-based subscription, may change and therefore the subscription aggregations need to be updated to reflect the current set of subscriptions in the system. This is a problem confronted in our previous work [5]. Consider a content-based router with a general subscription S. When S is to be removed, perhaps because the client is not interested in S any more, it becomes necessary to replace S with the more precise, but also more numerous, covered subscriptions. In this paper, we use $\mathbb{S}$ to denote subscriptions that are triggered by S, Since the subscriptions in $\mathbb{S}$ are no longer covered by any subscription, all the subscriptions in $\mathbb{S}$ need to be forwarded. To maintain correctness of the routing tables, all the subscriptions in $\mathbb{S}$ are forwarded immediately before removing S. As it is more costly to process subscriptions than other messages in covering-based routing [6], a large set of subscription messages can impose severe congestion in the network, overwhelming the brokers that must process this burst of subscription traffic, and destabilize the system. To illustrate the potential severity of this problem, Fig. 1 quantifies how broker queue lengths are affected by the burst of subscriptions that result from removing a small number of subscriptions. Here, the system only stabilizes after several hours. What's more, in systems with limited queue sizes, the burst of subscriptions will result in message loss.

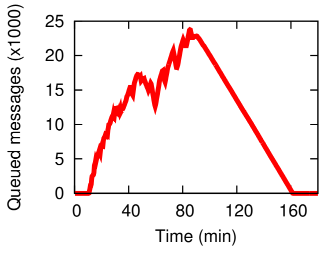


Fig. 1. Subscription bursts triggered by unsubscriptions can result in severe congestion that persists for a long time. (Setup detailed in Sec. V.)

The problem addressed in this paper may arise in any scenario with a high degree of subscription covering where some subscriptions are cancelled or modified. Here are some examples: (1) In a distributed workflow management system, each workflow activity issues subscriptions that reflect its narrow interest in the events that trigger it [7]–[10]. As well, an administrator wanting to monitor the entire system would issue a broad subscription S that covers many of the existing ones. When monitoring stops, the removal of the associated subscription S will result in a large and instantaneous burst of many subscriptions heretofore covered by S. (2) In a content delivery system [11]–[13], users may express both broad and narrow interests and share similar interests [14]. As a matter of fact, a general subscription might cover thousands of more specific subscriptions. For example, in a mobile content delivery system, drivers may subscribe to updates about traffic in their vicinity, while a traffic reporter needs to subscribe to all road conditions. When the reporter leaves or disconnects from the system, there is a risk of subscription bursts [15]. (3) In a financial system, analysts subscribe to updates about

This work is supported by the National Natural Science Foundation of China under Grant No.61070027, 61020106002 and 61161160566. Between 2008 to 2010, this work was supported by an NSERC Discovery Grant.

the market that is experiencing heavy activity. It is natural for some analysts' interests to be a superset of other analysts. As some analysts with broad interests leave, change interests, or disconnect from the system [15], there is a risk of subscription bursts. (4) In a distributed collaborative system [16], [17], participants issue subscriptions that reflect their interests. Due to their organizational roles, participants may have limited visibility in the system. For example, staff can only subscribe to events related to their own work, while a divisional manager is able to subscribe to all events that relate to the whole department. As the position of the divisional manager changes, there will inevitably be moments where a subscription that covers many others needs to be removed or modified, resulting in a subscription burst.

In the above scenarios, the removal of certain general subscriptions leaves the system susceptible to bursts of subscription propagations. A trivial solution to avoid the burst of covered subscriptions in $\mathbb{S}$ is to retain the covering subscription S even after a client has indicated it is not interested in S. However, this results in a situation where messages that satisfy S but not any subscriptions in $\mathbb{S}$ will be unnecessarily propagated by the routers. Depending on the message rates and the similarity of interest between S and $\mathbb{S}$, this *false positive* traffic can impose significant unnecessary network and processing load on the system. At the other extreme lie existing routing protocols [18], which immediately replace S with $\mathbb{S}$, thereby avoiding false positive message traffic but leaving the system susceptible to bursts of subscription propagations.

This paper navigates the solution space that lies between these two extreme possibilities. In particular, *portions* of the subscription propagation tree are selectively preserved or replaced with the more precise, but also more numerous, covered subscriptions. When a client removes a subscription S, a distributed algorithm is used in which each router determines whether it is better to preserve or remove S. The decision to preserve S depends on the similarity between S and the subscriptions that would replace it. If the similarity is high, there will be few needless false positive messages, and it is worthwhile to preserve S.

This paper makes the following contributions: In Sec. III, properties of subscription propagation in content-based routing networks are formalized. As well, a metric that uses message history to capture the similarity between subscriptions is developed. The metric and properties are then used in Sec. IV to develop a distributed, incremental filter aggregation algorithm that is composed of three parts: (1) Statistics collection operating at constant cost, (2) a similarity computation operating at linear cost, and (3) subscription tree pruning to incrementally prune the propagation tree of a subscription that is to be removed. Next, Sec. V presents a detailed evaluation of the performance of an implementation of the distributed pruning algorithm. A large number of experiments are conducted including ones based on real traces of a distributed workflow engine and a further workload extensively used in the literature. The results consistently show that the algorithms presented in this paper provide significant benefits by avoiding severe network congestion that persists for several hours, reduce routing table sizes by about 50%, and decrease message propagation delays by several orders of magnitude.

II. Related work

Content-based publish/subscribe routing: Content-based routing systems [3] have been developed for a variety of environments including dedicated acyclic overlays [3], [19], distributed hash tables [20]–[22] and wireless ad-hoc networks [23]. This paper focuses on systems such as Siena [3] and early versions of PADRES [24] that assume an acyclic overlay of dedicated brokers. In these systems, the information sources, or publishers, first advertise a description of the data they are about to send, and this advertisement message is flooded across the overlay and generates a spanning tree rooted at the publisher. Next, the data consumers, or subscribers, issue a subscription defining their interests, and this subscription is routed hop-by-hop along the reverse path of matching advertisement trees towards publishers. Finally, publications from publishers are disseminated hop-by-hop following the reverse paths of matching subscriptions until they are delivered to interested subscribers.

Covering optimization: Aggregating subscriptions and advertisements in a content-based network using a covering optimization was developed by Carzaniga [3], and has been implemented in many publish/subscribe systems. The covering relationship between filters can be depicted as: A filter F covers a filter G denoted by $F \supseteq G$, iff $N(F) \supseteq N(G)$, where $N(X)$ is the set of publications that match X. Based on this definition, similarities among subscriptions can be found and used to remove redundant subscriptions from the network, maintain compact routing tables and reduce network traffic.

In the PADRES system used in this paper, two variants of the covering optimization are implemented. The first one is called *active covering*, and strictly obeys the classic covering definition from [2], [3]. With active covering, when a subscription S' arrives at a broker after a subscription S that covers S', the broker does not forward S'. Moreover, if S' arrives before S, S is forwarded, and also brokers that see both S' and S delete S' from their routing tables. This helps to ensure more compact routing tables, but requires more processing and network traffic to clean up unnecessary S' routing state. Under *lazy covering* on the other hand, in the case where S' arrives before S, S is simply forwarded and none of the S' routing state is cleaned up. This is a cheaper operation but results in larger routing tables over time.

Although covering-based routing is good at aggregating subscriptions and maintaining a compact routing table and reducing network traffic, the experiments in our earlier work [5] show that the more the subscriptions are covered, the worse the system will behave when certain subscriptions are removed. This problem is especially severe for active covering, but is also present with lazy covering.

Merging optimization: The merging technique is used to further reduce the routing table size and the traffic overhead in the content-based network, and is used in addition to the covering optimization [3].

Formally, a filter F is a merger of a set of filters $F_1, ..., F_n$, iff $N(F) \supseteq (\bigcup_{i=1}^{n} N(F_i))$. There are two kinds of mergers. When the publication set of the merger is exactly equal to the union of the publication sets of the original filters, $N(F) = (\bigcup_{i=1}^{n} N(F_i))$, it is a *perfect merger*. Otherwise, if the publication set of the merger is larger than the union, it is an *imperfect merger*. Imperfect merging can reduce the number of subscriptions, but may allow publications to be forwarded that do not match any of the original subscriptions. In order to apply merging, it must be possible to efficiently compute mergers and if imperfect merging is performed the number of the unwanted publications must be small. While Crespo et al. [25] proposed merging of queries that are evaluated periodically against a database, they showed that in the general case query merging is NP-hard.

Congestion Control in publish/subscribe systems: Congestion control in pub/sub differs from techniques in other systems. Pietzuch *et al.* [26] presented a pub/sub congestion control scheme that combines two mechanisms: the subscriber host broker-driven protocol and publisher host broker-driven protocol. This solution adjusts the rate of publishing new messages, allowing brokers under recovery to eventually catch up, and other brokers to keep up. Although this paper handles the problem of congestion caused by publications, it does not address congestion triggered by unsubscriptions.

Aggregation and summarization techniques: Another class of techniques reduce the traffic in a pub/sub system by offering different publication delivery semantics. For example, aggregation operators, such as *average* and *sum*, can be added to subscriptions in order to compute a summary of publications [27], [28]. Similarly, *top-k* semantics can limit the set of delivered publications to a predictable amount of the most relevant publications in a given time window [29], [30]. These techniques are orthogonal to the approach in this paper, which preserves existing pub/sub semantics and addresses the congestion induced by subscription churn.

III. Subscription propagation properties

We now state some properties that arise in an acyclic content-based routing overlay, and also define a measure of similarity between subscriptions. These properties underlie the subscription pruning algorithm developed in Sec. IV.

A. Nature of subscription propagation

There are two types of subscriptions: *root* subscriptions which are not covered by any other subscription in the system and *non-root* subscriptions which are covered by some subscription in the system. Let S_r and S_n be a root and non-root subscription, respectively, and let T_{S_r} and T_{S_n} denote the subscription propagation trees of S_r and S_n.

Property 1. *Without the covering optimization, the propagation of any subscription S is a tree.*

Property 2. *With the covering optimization, T_{S_r} remains the same, and T_{S_n} can be reduced to a linear path.*

The proofs of the above properties are in Appendix A.

In the following discussion, the *publisher host broker* (PHB) and *subscriber host broker* (SHB) refer to the brokers to which a publisher and subscriber connect, respectively. Note that these are only logical designations, and any given broker may play the role of both a PHB and SHB.

B. Subscription similarity

Subscriptions filter out unwanted publications. When a root subscription S is removed, the subscriptions quenched by S, denoted as $\mathbb{S}$, must be activated to ensure that matched publications are forwarded accurately. However, if the filter function of S and that of $\mathbb{S}$ are the same, no benefit is gained by removing S and propagating $\mathbb{S}$. In other words, if the interests expressed by S and $\mathbb{S}$ are similar, keeping S instead of forwarding $\mathbb{S}$ results in a more compact routing table and avoids unnecessary messages.

However, it is difficult to quantify similarity using only subscription filters. First, determining the filter similarity between S and $\mathbb{S}$ involves computing a merger of $\mathbb{S}$, which is NP-hard [25]. Moreover, it may be inaccurate to quantify similarity using only the subscription filters. Suppose a root subscription S_1 indicates an interest for values in the range $[0, 100]$, and a covered subscription S_2 has an interest in values $[0, 50]$. Based only on these filters, the best one can assume is that S_2 will receive half as many publications as S_1. However, it may be that all publications have values in the range $[0, 50]$, and so S_1 and S_2 are practically identical in terms of how many publications they actually filter.

In addition, other subscriptions that also intersect S should affect our decision. For example, there may be another subscription S_3 with interest in the range $[40, 110]$. Notice that S_3 intersects S_1 but has no covering relationship with S_1. When S_1 is removed, there is no longer any filtering benefit in replacing S_1 with S_2, because $S_3 \cup S_1$ equals $S_3 \cup S_2$. In other words, those publications that are not of interest to S_2 need to be propagated anyway to be delivered to S_3.

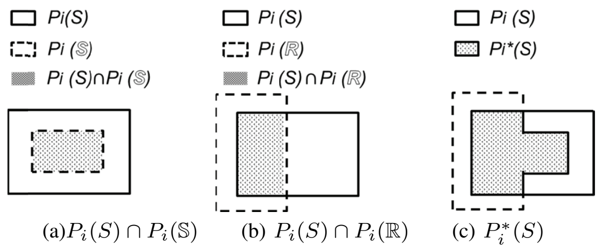


Fig. 2. Relationships among $P_i(S)$, $P_i(\mathbb{S})$, $P_i(\mathbb{R})$, and $P_i^*(S)$

For a given subscription S, let $\mathbb{R}$ refer to those subscriptions that intersect S but do not cover it or are covered by it. Let $P_i(\cdot)$ denote the set of publications matched by one or more subscriptions and let $P_i^*(S)$ denote the subset of $P_i(S)$ that satisfies the remaining set of subscriptions in the system. Fig. 2 shows the relationships among $P_i(S)$,

$P_i(\mathbb{S})$, $P_i(\mathbb{R})$, and $P_i^*(S)$. The shaded area in Fig. 2(a) is the intersection of $P_i(S)$ and $P_i(\mathbb{S})$, which contains publications that even after cancelling S, still need to be forwarded since they are of interest to subscriptions in $\mathbb{S}$. The shaded area in Fig. 2(b) is the intersection of $P_i(S)$ and $P_i(\mathbb{R})$, which contains publications that after cancelling S, also need to be forwarded due to the interests of subscriptions in $\mathbb{R}$. The shaded area in Fig. 2(c) represents publications that need to be forwarded after cancelling S since they are of interest to other subscriptions in the system. From Fig. 2, we can see that $P_i^*(S)$ must either match $\mathbb{S}$ or match $\mathbb{R}$. Hence, we obtain a metric to calculate $P_i^*(S)$:

$$P_i^*(S) = P_i(S) \cap (P_i(\mathbb{S}) \cup P_i(\mathbb{R})). \tag{1}$$

Then, the similarity, ϕ, between S and $\mathbb{S}$ is given by:

$$\phi = \frac{|P_i^*(S)|}{|P_i(S)|} \qquad \phi \in [0, 1]. \tag{2}$$

C. Non-decreasing similarity

Note that the similarity, ϕ, between S and $\mathbb{S}$ differs at every broker. As it can be expensive to record the history of publications and compute the similarity at each broker, we prove an important property in this section that allows us to compute similarity more cheaply.

Recall that for a given publisher P, a matched root subscription S must be propagated along a path from the SHB of S to the PHB of P (see Property. 1). It turns out that along this path, the similarity metric defined in Sec. III-B is non-decreasing. This property, which is defined below and proved in Appendix B, is exploited in Sec. IV to incrementally prune portions of a subscription's propagation tree.

Property 3. *For root subscription S and a matching publisher P, the similarity, ϕ, is non-decreasing at each broker on the path from the SHB of S to the PHB of P.*

IV. Incremental filter aggregation algorithm

Let $unsub(S)$ denote the unsubscription message of S. According to Property 2, for every publisher P, in the propagation path of $unsub(S)$, the similarity between S and $\mathbb{S}$ increases. Since $unsub(S)$ has the same propagation path as S, if we choose not to propagate the $unsub(S)$ message, the subscription S will serve to aggregate the interests of $\mathbb{S}$ (since S covers $\mathbb{S}$). The objective of our incremental filter aggregation algorithm is to find a suitable broker, where the similarity ϕ is large enough to stop forwarding the $unsub(S)$ message.

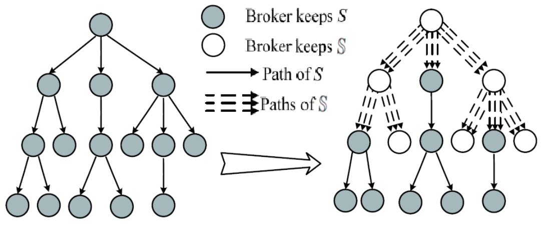


Fig. 3. Illustration of incremental filter aggregation

Fig. 3 depicts the outcome of applying the incremental filter aggregation algorithm. The brokers in Fig. 3 form the propagation path of S. In particular, the white brokers are those where S has been replaced with the more precise, but also more numerous, subscriptions in $\mathbb{S}$, and the shaded brokers are those where S has been selectively preserved.

Our incremental filter aggregation algorithm has the following properties:

1) With a given similarity threshold, for any publisher P, there exists a *critical broker* between the SHB of S and the PHB of P, after which the similarity between S and $\mathbb{S}$ is larger than the threshold.
2) The similarity threshold is effected by the size of $\mathbb{S}$. If the size of $\mathbb{S}$ is small, a broker tends to use a larger threshold, otherwise, a broker will choose a smaller threshold so as to avoid bursts of subscription traffic.
3) If a broker is congested, unsubscription messages should be blocked in that broker, and re-activate unsubscription processing when congestion subsides.

Computing the similarity between S and $\mathbb{S}$ hop by hop on the propagation path of $unsub(S)$ is computationally expensive. To determine the critical broker with small overhead in a distributed manner, we employ three techniques to reduce computing cost:

1) For any subscription S, information about its historical publications are only recorded at the broker to which the subscriber connects to, i.e., the SHB of S.
2) For a give similarity threshold, it is sufficient to use the statistics recorded in the SHB to infer the position of the corresponding *critical broker*.
3) In the propagation path of $unsub(S)$, the broker can choose a suitable similarity threshold by the size of $\mathbb{S}$, so as to find a suitable *critical broker* to stop $unsub(S)$.

Our incremental filter aggregation algorithm comprises three parts: *statistics collection*, *critical broker computation* and *subscription tree pruning*.

A. Statistics collection

The similarity between S and $\mathbb{S}$ and the size of $\mathbb{S}$ are core factors in the incremental filter aggregation algorithm. The latter can be visualized with the propagation of $unsub(S)$, and the former can be inferred from the publication matching results. To collect the necessary statistics, a value we refer to as *distance* is added to the publication message header, which records the number of overlay hops between the SHB of subscription S and the nearest broker with another subscription S' that also matches the publication.

Definition 1. *For a publication P_i and subscription S, let B_s be the SHB of S, B_p be the PHB of P_i, and B_n be the nearest broker on the path between B_s and B_p that maintains another subscription that also matches P_i. Then,* distance *is the number of overlay hops between B_s and B_n.*

A PHB initializes a publication's *distance* to zero. Then, at every other broker if the publication matches multiple

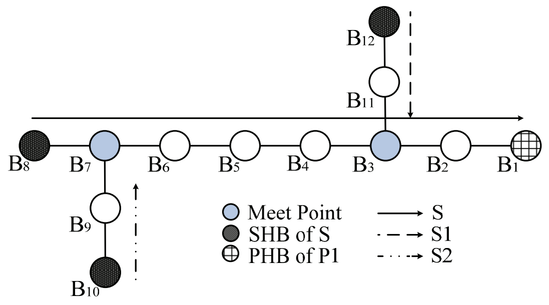


Fig. 4. Publication propagation example

subscriptions, *distance* is reset to zero; otherwise, *distance* is incremented by one in the publication.

The cost of statistics collection is negligible. For the example in Fig. 4, Broker B_1 receives a publication P_1 from a publisher, and P_1 matches both S, S_1 and S_2, where S covers S_1 and S_2. At first, B_1 initializes *distance* of P_1 to zero, and at broker B_2, *distance* of P_1 is incremented by one. At Broker B_3, P_1 matches both S and S_1, so *distance* of P_1 is reset to zero before being forwarded to Brokers B_4 and B_{11}. This process is repeated until P_1 arrives at Broker B_6 with a *distance* value of 3. At Broker B_7, the publication matches S and S_2, so *distance* is reset to zero again. Finally, P_1 arrives at B_8 with *distance* equal to 1. From this, Broker B_8 deduces that the nearest broker with a subscription that is also interested in P_1 is one hop.

pubID	publisherID	subID	distance
P_1	P	S	1
P_2	P	S	5
P_3	P	S	1
P_4	P	S	1
P_5	P	S	7

TABLE I
PART OF DATABASE AT BROKER B_8 FROM FIG. 4

Each broker that is the SHB for a subscription that matches a publication records the publication identifier, publisher identifier, subscription identifier, and the *distance* value of the publication. Note that the publication is not stored, keeping the database footprint small. Table I shows what the database at Broker B_8 might look like. The entries in the database can expire after a configurable time so that they maintain information about a sliding window of matching publications. Furthermore, these entries are purged when the associated subscriptions are unsubscribed.

B. Critical broker computation

Based on a given similarity threshold, for each publisher, the SHB can efficiently and locally determine the distance to the *critical broker* using the $distance$ value. The distance from the SHB to the *critical broker* is called the *critical distance*. For a given publisher P, $\{P_i\}$ denotes all publications that match both S and P in the database, and $\{P_i(\mu)\}$ denotes publications that not only match both S and P but also have a *distance* smaller or equal to μ. Let $B(\mu)$ denote a broker whose distance from the SHB of S in the path from SHB of S to PHB of P is μ, and use $\phi(\mu)$ to represent the similarity value ϕ of this broker, then:

$$\phi(\mu) = \frac{|P_i(\mu)|}{|P_i|} \qquad \phi(\mu) \in [0,1] \tag{3}$$

Proof: For any publication P_i, if $P_i.distance$ is less than μ, then according to the definition of *distance*, $B(\mu)$ still receives P_i after cancelling S. Hence, in broker $B(\mu)$, $\{P_i(\mu)\} = P_i^*(S)$. As the SHB of S records all the publications matching S, $\{P_i\} = P_i(S)$. According to our definition $\phi = \frac{|P_i^*(S)|}{|P_i(S)|}$ (see Equation 1), since $\{P_i(\mu)\} = P_i^*(S)$ and $\{P_i\} = P_i(S)$, in broker $B(\mu)$, $\phi = \frac{|P_i(\mu)|}{|P_i|}$. ■

Suppose that subscription S is to be unsubscribed. Table I gives us the similarity threshold for S at every broker. Consider a broker that is two hops away in P's propagation path. At this broker, $\{P_i\} = \{P_1, P_2, P_3, P_4, P_5\}$ and $\{P_i(2)\} = \{P_1, P_3, P_4\}$. Consequently, at this broker, S's similarity threshold is 0.6.

Alternatively, given a similarity threshold, we can find the critical broker for each publisher. For instance, for a similarity threshold of 0.8, the critical broker is x hops on the path from the SHB to the PHB, where 80% of publications have $distance$ values less than or equal to x. For the example in Table I, for publisher P, the value of $x = 5$. According to our definition, if $\phi = 80\%$ the *critical distance* for P is 5.

C. Subscription tree pruning

When we prune a subscription tree, it is important to consider not only the similarity threshold, but also the number of triggered subscriptions. The SHB records the first measure in the header of the unsubscription, and each broker can easily compute the second measure since it would be the one forwarding the triggered subscriptions anyway. At brokers with a different number of triggered messages, M, we use different similarity thresholds to compute whether a subscription needs to be pruned. We use the following formula in our experiments in Sec. V: $\phi = 60\%$ when $lg(M) \geq 3.7$, $\phi = 90\%$ when $3.7 > lg(M) \geq 1$, and $\phi = 100\%$ when $1 > lg(M)$.

Here, when the number of triggered messages, M, is less than 100, we do not consider pruning subscriptions, whereas if M is in the range $[100, 5000]$, the similarity threshold is 90%, and when $M > 5000$, the threshold is reduced to 60%.

The algorithm has two steps. First, we find the critical distances to each publisher for each similarity threshold. For example, in Table I, the distance for ϕ_1 is 1, the distance for ϕ_2 is 10, and we do not need to compute the distance for ϕ_3 because a threshold of 100% requires the subscription to always be removed. Next, a list of tuples is added to the unsubscription message with each tuple containing three values: the identifiers of matched publishers (*publisherID*), and critical distances (*pruneDis1* for ϕ_1, and *pruneDis2* for ϕ_2).

Each broker uses Algorithm 1 to determine whether to continue forwarding the unsubscription and prune the next hop of the subscription tree. The time complexity of Algorithm 1 is $O(N * P)$, where N denotes the number of neighbours, and P is the number of matched publishers.

Algorithm 1: Handler of *Unsubscription Message*

```
Input: $us \leftarrow$ an unsubscription message
if $lg(|\mathbb{S}|) \geq 3.7$ then
    pruneDis=pruneDis1;
if $3.7 > lg(|\mathbb{S}|) \geq 1$ then
    pruneDis=pruneDis2;
if $1 > lg(|\mathbb{S}|)$ then
    pruneDis=$+\infty$;
forall the neighbour $U$ of current broker do
    generate a new tuplelist $newlist$;
    $newlist \leftarrow \emptyset$;
    forall the tuples
    $\{publisherID, pruneDis1, pruneDis2\} \in unsub.tuplelist$ do
        if $publisherID.sender = U$ and $pruneDis > 0$ then
            $newlist \leftarrow newlist \cup \{publisherID, pruneDis1 --$
            $, pruneDis2 --\}$;
    if $newlist \neq \emptyset$ then
        generate a new Unsubscription Message $nus$;
        $nus \leftarrow us$;
        $nus.tuplelist \leftarrow newlist$;
        sent $nus$ to $U$;
```

Ideally, for every neighbour U of the current broker, if all the publishers from U have reached its critical broker, then it is unnecessary to deliver the unsubscription message to U. In other words, the remainder of the subscription tree is preserved. Moreover, note that in a congested network, the unsubscription that is triggering many subscriptions will naturally be delayed, even before it reaches its critical broker, and will be processed when congestion subsides.

V. Evaluation

Our evaluations consider three very different scenarios and workloads: our own controlled configuration and workload, a mobile scenario using an existing workload (also employed by related research [31]), and a distributed business process monitoring scenario using a workflow engine workload.

The protocols in this paper have been implemented in the PADRES content-based pub/sub prototype.[1] The experiments are run on a cluster of 21 machines each with four 1.86 GHz Xeon processors and 4 GB of RAM. This setup mimics a data center environment and offers a controlled system from which we can derive meaningful analysis.

We compare the incremental pruning algorithm proposed in this paper with three other algorithms described below.

Active covering: The traditional covering algorithm. When a broker encounters a new subscription S that covers existing uncovered subscriptions $\mathbb{S}$ from the same neighbour, it forwards S, then forwards unsubscriptions for $\mathbb{S}$ and then deletes $\mathbb{S}$ from its routing table to keep the routing tables compact. Similarly, when S is unsubscribed, it must again forward all the subscriptions $\mathbb{S}$ to ensure publications are routed correctly.

Lazy covering: When a new subscription S covers existing uncovered subscriptions $\mathbb{S}$, the broker simply forwards S, and when S needs to be unsubscribed, only subscriptions that arrive after S may need to be forwarded. This algorithm forwards fewer subscriptions, but routing tables grow over time impacting publication matching and delivery performance.

Subscription packing: An optimization of the active covering algorithm: subscriptions or unsubscriptions $\mathbb{S}$ that need to be forwarded at once are instead forwarded in one message. This reduces the number of messages sent, and utilizes the knowledge that these messages have a covering relationship to more efficiently index them in the covering data structure. However, there is no benefit in matching and forwarding these subscriptions as they must be processed individually. The details of the subscription packing algorithm are presented in an extended version of this paper [32].

We quantify performance with the following metrics.

Publication propagation delay: The end-to-end latency of publications as they traverse from a publisher to a subscriber.

Input queue size: The number of messages waiting to be processed by a broker. In the experiments, the queue size is used as a measure of congestion in the network, and shows how quickly the system stabilizes after unsubscribe operations.

Routing table size: The number of subscriptions stored in the routing table of a broker. It also reflects the amount of transmitted subscription messages. Smaller routing tables consume less memory and result in faster routing computations.

Triggered messages: The number of triggered subscription messages at the different brokers.

False positive publications: A count of the number of publications that are unnecessarily forwarded towards uninterested subscribers. Only the incremental algorithm has false positives.

A. Controlled scenario

The network topology consists of 49 brokers, as shown in Fig. 5, with the five brokers labelled A–E denoted as core brokers. Publishers connect to the 3 edge brokers close to broker E, and subscribers are randomly distributed among the remaining edge brokers.

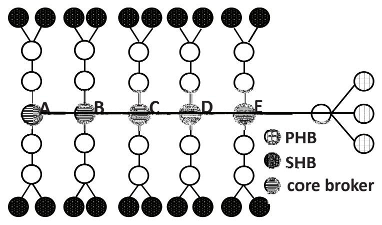
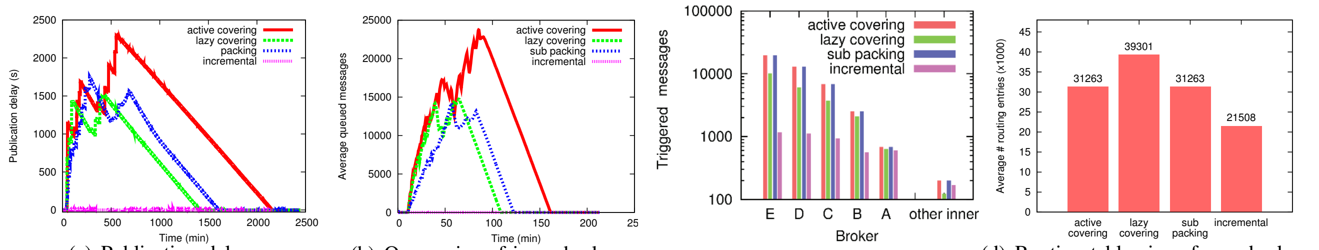


Fig. 5. Default broker topology

While this topology has been chosen to run controlled experiments, it also mimics a typical distributed messaging platform that drives workflow processing in large enterprises with multiple departments and external supply chain partners. Enterprise components and services involved in the workflow subscribe to event triggers and are invoked through messages from external sources [7]–[10].

1) Effect of covering: Before studying how unsubscriptions can induce congestion, we first present the benefits of the covering algorithms in the absence of unsubscriptions. In this experiment, we issue 42 000 subscriptions and 200 000 publications and measure how long brokers spend processing subscriptions (split into the covering detection and matching phases), unsubscriptions, and publications. In Table II, we see that the active covering algorithm expends more cycles detecting covering relationships, but can process publications in half the time. Therefore, active covering is a good choice when the subscription set is relatively stable and publication delivery

[1]Available at http://padres.msrg.toronto.edu.



(a) Publication delay (b) Queue size of inner brokers (c) Triggered messages at brokers (d) Routing table size of core brokers

Fig. 6. Comparisons of active covering, lazy covering, subscription packing, and incremental unsubscription algorithms

	Active		Lazy		None	
Sub matching	14.5	(0.81)	28.8	(1.6)	33.7	(1.6)
Sub covering	216	(12)	486	(27)	0	(0)
Unsub	11.7	(1.4)	0	(0)	0	(0)
Pub	**600**	(3.0)	**1240**	(6.2)	**1580**	(7.9)
Total	842		1754		1613.7	

TABLE II
COMPARISON OF TOTAL BROKER PROCESSING TIME (IN SECONDS) FOR DIFFERENT MESSAGE TYPES UNDER ACTIVE COVERING, LAZY COVERING, AND NO COVERING ALGORITHMS. (THE AVERAGE PROCESSING TIME PER MESSAGE (IN MILLISECONDS) IS SHOWN IN PARENTHESES.)

latency is important. As there are no unsubscriptions issued in this workload, we do not see the subscription processing benefits of lazy covering, but we do note that its publication processing delay is larger than with active covering, and will only get worse with subscription churn.

We further emphasize that although subscription covering can reduce the publication matching time, the cost of maintaining the covering relationships among subscriptions is also expensive [6]. For example, Table II shows that this covering computation is an order of magnitude more expensive than subscription matching and dominates the subscription processing pipeline. Since an unsubscription can trigger many subscriptions, the covering computation of those triggered subscriptions is the main cause of the system congestion.

The remainder of the experiments will show how the incremental algorithm can achieve the low publication processing times of the active covering algorithm without the associated unsubscription induced congestion.

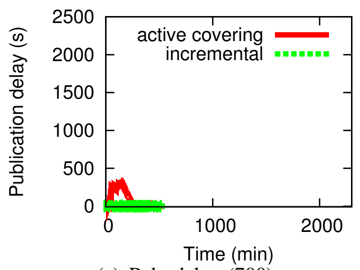
2) Comparisons among algorithms: The first set of experiments compare the incremental algorithm proposed in this paper with the traditional active covering algorithm, as well as the lazy covering, and packing algorithms. In this group of experiments, 80 000 subscriptions are issued after which we trigger 20 unsubscriptions, one every 0.5 s. After these unsubscriptions are sent, the experiment is allowed to run for several hours to measure how the algorithms process any resulting message bursts and congestion. In the workload, the average number of triggered subscriptions for each unsubscription is 2 100, and the similarity threshold for the incremental algorithm is set at 90%, which implies a 10% publication false positive rate. As well, to make the experiments more realistic, publishers send a continuous stream of background publications every 10 ms.

First, we examine the propagation delay of publications. Fig. 6(a) plots how the propagation delay for the four algorithms varies over time. The active covering algorithm suffers from peak delays of over 2 400 s with average delays of over 1 000 s during the experiment. The lazy covering and subscription packing algorithms deliver publications in about 60% of this time on average. The incremental unsubscription algorithm, however, performs substantially better, with average delays of only about 0.4 s, an improvement of more than 99%. Moreover, congestion subsides quickly with the incremental algorithm, and the average steady state publication delay falls to 200 ms. Next, we will investigate the causes behind these publication delays.

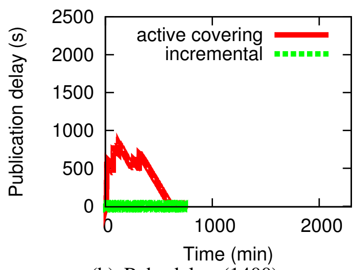
Fig. 6(c) plots (on a log scale) the number of triggered subscription messages ($\mathbb{S}$) at the core brokers A–E, and shows that with active and lazy covering, as well as subscription packing, $\mathbb{S}$ increases rapidly at brokers approaching the publisher, which in this case is broker E. For example, with active covering, there are about 700 messages triggered at broker A increasing to almost 20 000 messages at broker E. With the incremental algorithm, many triggered subscriptions are forwarded no further than the critical broker, and consequently only about 1 100 triggered messages arrive at broker E. Fig. 6(c) also confirms the claim made in Sec. III-C that the incremental algorithm is more inclined to preserve subscription trees at brokers closer to a publisher. Finally, the results show that the number of triggered messages at the remaining inner brokers is much less than that at the core brokers.

As most of the congestion occurs at the core brokers, we now focus on how the input queue of these brokers are affected by the messages triggered by unsubscriptions. As shown in Fig. 6(b), the queue size of the core brokers explodes to almost 24 000 messages with the active covering algorithm. Queue sizes in the lazy covering and packing algorithms are smaller but still experience a large burst of up to 15 000 messages. However, the incremental algorithm far outperforms all of these with only a slight increase in the input queue size peaking at 150 messages. More importantly, the incremental algorithm stabilizes very quickly, while the active and lazy covering only manage to slightly reduce the queue size even after one hour and a half into the experiments. These large long-lived queues indicate that the system is overloaded and cannot function normally. Seeing how the delays in Fig. 6(a) track the queue size in 6(b), it is evident that the delays are a result of the congested queues.

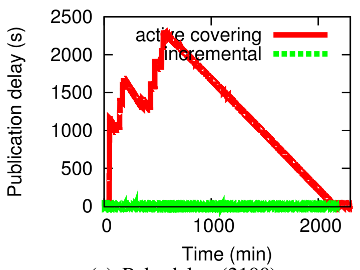
Furthermore, the large publication delays in Fig. 6(a) are consistent with queuing theory results which show that queuing delays increase rapidly as the message arrival rates



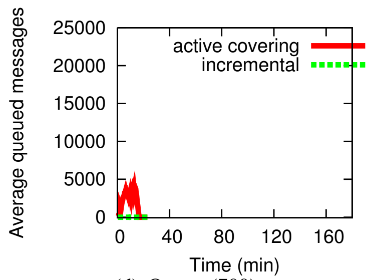
(a) Pub. delay (700)



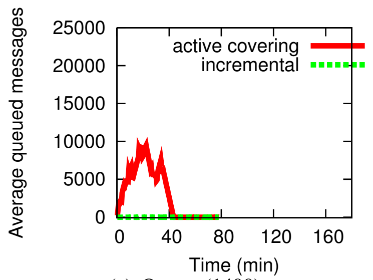
(b) Pub. delay (1400)



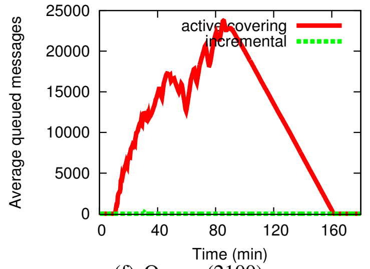
(c) Pub. delay (2100)



(d) Queue (700)



(e) Queue (1400)



(f) Queue (2100)

Fig. 7. Varying the number of triggered subscriptions

approach the link capacity [33].

It should also be noted that the burst of subscriptions in $\mathbb{S}$ triggered by an unsubscription of S does not only occur when the unsubscription is first issued. Instead, as the unsubscription makes its way through the network, it may trigger, at each broker along the unsubscription's propagation tree, the immediate forwarding of the subset of $\mathbb{S}$ that exists at that broker. This explains why the queue lengths in Fig. 6(b) fluctuate; as an unsubscription propagates slowly through the network, it will sometimes arrive at a broker with many covered subscriptions and cause these subscriptions to be forwarded thereby giving rise to another subscription burst. Moreover, these triggered subscriptions themselves may affect the covering relationships at the downstream brokers, causing perhaps even more congestion.

The subscription packing algorithm decreases the number of triggered messages by including them within a single message envelope, and tries to insert all the triggered subscriptions as a whole in the covering data structure. However compared to the incremental algorithm, the benefits of packing are limited. Fig. 6(b) demonstrates that the network can still be severely congested with thousands of publications and subscriptions pending in the brokers' queues.

In addition to triggering fewer messages, the incremental algorithm also resulted in smaller routing table sizes in this experiment. Fig. 6(d) plots the average routing table size at the core brokers at the end of the experiment after the unsubscriptions have been issued. We see that while the active covering algorithm achieves compact routing tables, the lazy covering algorithm, which relies on the order that subscriptions are issued and therefore only opportunistically benefits from covering relationships, causes larger routing tables. However, the incremental algorithm most aggressively aggregates subscriptions and achieves smaller routing table size for it gives more opportunities for filter aggregation. It should be noted that the routing table size benefits of the incremental algorithm only manifest when covering subscriptions are unsubscribed, so it may not always outperform the active covering algorithm.

One overhead of the incremental algorithm is the gathering of statistics on a window of publications. Recall, however, that only the subscriber hosting brokers are required to maintain this information. In this experiment this information was stored in an Apache Derby database, the largest of which contained about 28 000 entries requiring about 24 MB of storage. The average broker's database was an even more modest 12 MB. Note that the databases only store the publication IDs rather than the entire publication messages, and we expire old publications, keeping the database small. More importantly, the results presented above on a real running implementation of the protocol show that, in spite of any overhead, the incremental algorithm far outperforms the traditional ones.

3) Varying the number of triggered subscriptions: This section investigates the effects of varying the number of trigger messages from 700 to 2 100. Since the results in Sec. V-A2 showed that the performance of the active covering, lazy covering and packing algorithms are similar, here, we compare the incremental algorithm with only the active covering algorithm—the most widely used one. The default number of unsubscriptions is 20.

Beginning again with the publication delays, Figs 7(a), 7(b), and 7(c) show the active covering algorithm is sensitive to the number of triggered subscriptions but the incremental algorithm provides relatively stable performance. Moreover, the results indicate a wide distribution in the publication delays under the active covering algorithm.

Our results show that after unsubscribe operations, the routing table sizes sharply increase with increasing triggered subscription workloads: from just under 8 000 to over 21 000 for the active covering algorithm. More interestingly, since more triggered subscriptions give more opportunities for filter aggregation, the incremental routing table sizes decrease with increasing number of triggered subscription workloads. All in all, the benefits of the incremental algorithm accrue as the number of triggered subscriptions increases.

Figs. 7(d), 7(e), and 7(f), show that workloads that exhibit more covering relationships among subscriptions are more likely to congest the system resulting in large queue lengths. This is understandable since more covering relationships are positively correlated with larger $\mathbb{S}$ sets, which are prone to lead to subscription bursts when the root subscription S is removed. For example, when the number of triggered subscriptions is 10, the active covering and incremental algorithm both result in relatively minor and short-lived congestion. When the number of triggered subscriptions increases to 1 400, the active covering algorithm, in Fig. 7(e), experiences bursty traffic with queue lengths of about 10 000 messages. This trend continues with active covering performing worse with a larger number of triggered subscriptions; when the average number of triggered subscriptions is 2 100, the queue sizes continue to increase an hour into the experiment and remain even after two hours. Contrast this behaviour to the incremental algorithm, in Fig. 7(f), which only experiences a short-lived congestion of fewer than 150 messages.

B. Mobile scenario

In this group of experiments, 17 brokers connected in a star topology, as in Fig. 8, support clients moving about a city. There are 2 000 publishers, each of whom publish once every 1 s to 15 s, and 200 000 mobile subscribers, perhaps receiving publications on their mobile phone.

We focus on the rush hours, from 8:00 AM to 10:00 AM and 5:00 PM to 7:00 PM. During these periods, 60% of the subscribers are commuters and move from home to their office in the morning and back home in the afternoon, 20% of the subscribers remain stationary, and the remaining 20% move randomly according to a normal distribution. We use the message workload employed to evaluate the SIENA publish/subscribe system [31]: there are 200 000 subscriptions, each with 1 to 10 constraints, the attribute names of which are selected from 1 000 words according to a Zipf distribution; the distributions of attribute operations are 60% equality, 20% less-than, and 20% greater-than; and attribute values are chosen randomly from among 100 numbers. Publications have between 1 and 19 constraints, and their values are also chosen randomly from among 100 numbers.

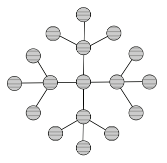


Fig. 8. Topology

We only present the results for the rush hour period (7 AM to 9 AM), when many subscribers are moving. Fig. 9(a) shows that publication delivery delays increased sharply, and in the worst case, a publication was delayed for more than 1 200 seconds with active covering, a delay that may be unacceptable in many applications. However, with the incremental algorithm, the longest publication delivery time is 17 seconds.

Turning our attention now to the queue size in Fig. 9(b), we see that the delays above were caused by congestion triggered by the periodic unsubscription messages. Further analysis showed that there are about 1 600 root subscriptions among the 200 000 subscriptions, each covering more than 2 000 subscriptions. (Note that a subscription may be covered by multiple root subscriptions.) Removing and reinserting those root subscriptions is very expensive. With the incremental algorithm, most of the root subscriptions' trees are preserved, and non-root subscriptions only have very short propagation paths (as explained in Sec. III).

Fig. 9(c) shows how the routing table size varies with time. With active covering, there is considerable subscription churn, but the overall table sizes are stable. The covering algorithm only benefits when the root subscription covers subscriptions that arrived from the same last hop; a root subscription that propagates from broker A to broker B cannot quench a subscription propagating from B to A. On the other hand, since the incremental algorithm preserves "old" subscription paths, a root subscription that moves a lot may leave behind several paths and thereby cover more subscriptions. This explains the decreasing routing table size in Fig. 9(c).

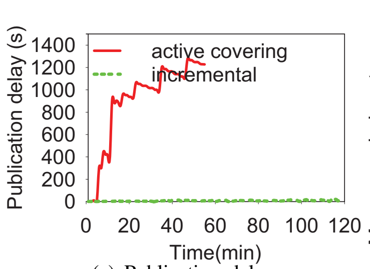
C. Distributed workflow scenario

In this experiment we evaluate a distributed business process execution engine built over a publish/subscribe overlay. The broker topology consists of four domains, each representing a business partner, and each domain contains ten brokers arranged in a three level hierarchy. Within each domain are six process execution engines that can host and execute activities within a process [7].

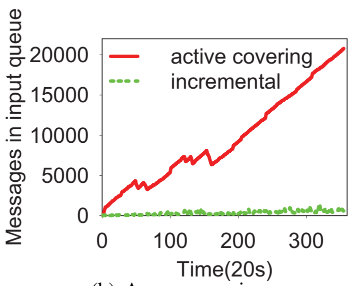
Three classes of business processes are deployed across these execution engines. A small process contains three activities all of which are deployed to a single randomly chosen engine; a medium process contains ten activities which are deployed across the engines within a randomly chosen domain; and a large process has 26 activities each of which are deployed to a random engine in the system. The latter represent processes that involve multiple business partners. There are 80 small, 80 medium, and 40 large processes for a total of 200 processes. These processes contain combinations of parallel and sequential control flows, as well as loops [7]. Once deployed, these processes are periodically triggered and the associated engines issue advertisements, subscriptions, and publications to coordinate the flow of each process instance.

In addition, 30% of the processes are monitored by a client that subscribes to the associated control flow messages. This could represent a developer monitoring a process, or a business analyst tracking certain metrics about the process. Furthermore, four clients periodically subscribe to all process trigger messages for a period of one minute every ten minutes. This may be useful to an administrator sampling the system in order to profile it, or an auditor randomly inspecting certain process behaviour.

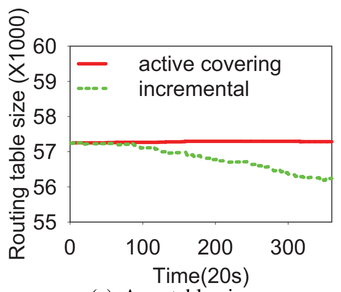
Fig. 10(a) shows a periodic spike in publication delivery delays with active covering that corresponds to when a moni-



(a) Publication delay

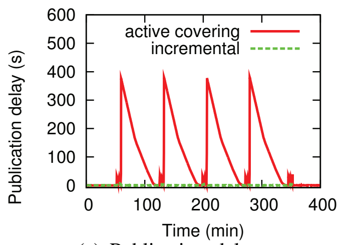


(b) Avg. queue size

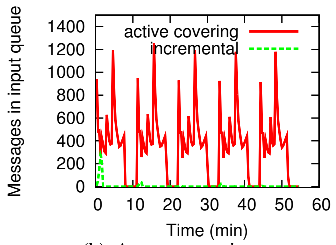


(c) Avg. table size

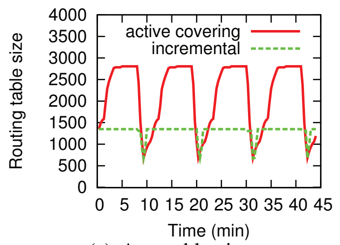
Fig. 9. Mobile scenario



(a) Publication delay



(b) Avg. queue size



(c) Avg. table size

Fig. 10. Distributed business process scenario

toring client stops monitoring and issues an unsubscription. In the worst case, the end-to-end delay for a publication is 400 s. The incremental algorithm, however, maintains stable delays, averaging about 0.24 s.

Turning our attention now to the average queue size in Fig. 10(b), we see that the cause for the above delays is due to congestion triggered by the periodic unsubscription messages. The maximum average queue size is about 1 200 but in the core brokers, the maximum is above 18 400 and is the reason why some publications encounter delays of 400 s. Again, however, the incremental algorithm suffers virtually no congestion.

Furthermore, Fig. 10(c) shows how the average routing tables experience large fluctuations as the active covering algorithm repeatedly attempts to optimize the entries in the table. The incremental algorithm also reacts to changing subscriptions but the modifications to the routing tables are much less severe.

VI. Conclusions

This paper addresses a neglected and perhaps largely unknown problem of severe congestion triggered by the removal of subscriptions in a content-based routing network. In particular, the popular covering optimization which aggregates a set of subscriptions with a single subscription S, is susceptible to subscription bursts, when S is unsubscribed.

We propose a light-weight and fully distributed congestion control algorithm. Instead of tearing down an entire subscription propagation tree when an unsubscription is issued, the algorithm incrementally prunes portions of the tree. The pruning decisions are made locally by each broker which considers both the congestion that would be triggered by removing the aggregate covering subscription and forwarding the replacement covered subscriptions, as well as the false positive publications that would arrive if the subscription were not removed.

The proposed incremental covering algorithm exploits a number of properties of subscription propagation in a content-based network that are observed and proved in this paper. In addition, it uses a computationally cheap yet accurate formula to compute the similarity among a set of subscriptions. The similarity calculation is based on the actual history of publications seen and hence gives a more accurate measure of the “effective” similarity for a given workload.

Real quantitative comparisons of implementations of the proposed incremental covering algorithm with the traditional ones reveal severe congestion under the traditional algorithms that are virtually eliminated with the incremental algorithm. In addition to sharply reducing the network congestion, the proposed algorithm can provide much smaller routing tables and thereby faster broker matching operations.

We conclude from the results that the traditional covering algorithms are not suitable for applications that may experience subscription churn, and in particular a large number of unsubscriptions. The congestion that results in such scenarios is excessive, and persist for an extended period of time. For example, in some experiments, queue lengths in the thousands of messages hardly diminished even after almost two hours. The proposed incremental algorithm, on the other hand, experienced virtually no congestion.

We are encouraged by the significant benefits of the work proposed in this paper, and plan to improve and investigate it further. One avenue for future work is to adapt the proposed mechanism to dynamic environments with mobile publishers and subscribers and changing workloads, such as a new subscription operation subsequent to the unsubscription operation. We also plan to construct additional techniques to reduce the number of false positive messages.

References

[1] J. Lawrence, "Designing multiprotocol label switching networks," 2001.

[2] G. Mühl, "Large-scale content-based publish-subscribe systems," Ph.D. dissertation, Technische Universität Darmstadt, Sep. 2002.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM ToCS*, 2001.

[4] 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.

[5] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen, "Transactional mobility in distributed content-based publish/subscribe systems," in *IEEE ICDCS*, 2009.

[6] S. Tarkoma and J. Kangasharju, "Optimizing content-based routers: posets and forests," *Distributed Computing*, vol. 19, pp. 62–77, 2006.

[7] G. Li, V. Muthusamy, and H.-A. Jacobsen, "A distributed service-oriented architecture for business process execution," *ACM Trans. Web*, 2010.

[8] D. Wodtke, J. Weisenfels, G. Weikum, and A. K. Dittrich, "The Mentor project: Steps toward enterprise-wide workflow management," in *ICDE*, 1996.

[9] P. Muth, D. Wodtke, J. Weisenfels, A. K. Dittrich, and G. Weikum, "From centralized workflow specification to distributed workflow execution," *JII*, vol. 10, no. 2, pp. 159–184, 1998.

[10] M. Sadoghi, M. Jergler, H.-A. Jacobsen, R. Hull, and R. Vaculín, "Safe distribution and parallel execution of data-centric workflows over the publish/subscribe abstraction," *IEEE TKDE*, 2015.

[11] I. Podnar, M. Hauswirth, and M. Jazayeri, "Mobile push: Delivering content to mobile users," in *DEBS*. IEEE Computer Society, 2002.

[12] I. Koenig, "Event processing as a core capability of your content distribution fabric," in *Gartner Event Processing Summit*, 2007.

[13] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh, "Cobra: Content-based filtering and aggregation of blogs and RSS feeds," in *NSDI*, 2007.

[14] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *IEEE INFOCOM*, 1999.

[15] D. Conan, S. Chabridon, and G. Bernard, "Disconnected operations in mobile environments," in *IPDPS*, 2002.

[16] P. Fenkam, E. Kirda, S. Dustdar, H. Gall, and G. Reif, "Evaluation of a publish/subscribe system for collaborative and mobile working," in *WETICE 02*, 2002.

[17] C. Canas, K. Zhang, B. Kemme, J. Kienzle, and H.-A. Jacobsen, "Publish/Subscribe Network Designs for Multiplayer Games," in *Middleware*, 2014.

[18] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman, "An efficient multicast protocol for content-based publish-subscribe systems," in *ICDCS*, 1999.

[19] V. Muthusamy, M. Petrovic, and H.-A. Jacobsen, "Effects of routing computations in content-based routing networks with mobile data sources," in *MOBICOM*, 2005.

[20] P. R. Pietzuch and J. Bacon, "Peer-to-peer overlay broker networks in an event-based middleware," in *DEBS*, 2003.

[21] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *ICDSP*, 2001.

[22] V. Muthusamy and H.-A. Jacobsen, "Infrastructure Free Content-Based Publish/Subscribe," *ACM/IEEE Trans. on Networking*, November 2013.

[23] M. Petrovic, V. Muthusamy, and H.-A. Jacobsen, "Content-based routing in mobile ad hoc networks," in *MobiQuitous*, July 2005.

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

[25] A. Crespo, O. Buyukkokten, and H. Garcia-Molina, "Query merging: Improving query subscription processing in a multicast environment," *IEEE TKDE*, 2003.

[26] P. R. Pietzuch and S. Bhola, "Congestion control in a reliable scalable message-oriented middleware," in *Middleware*, 2003.

[27] J. Chen, L. Ramaswamy, and D. Lowenthal, "Towards efficient event aggregation in a decentralized publish-subscribe system," in *DEBS*, 2009.

[28] N. K. Pandey, K. Zhang, S. Weiss, H.-A. Jacobsen, and R. Vitenberg, "Minimizing the Communication Cost of Aggregation in Publish/Subscribe Systems," in *ICDCS*, 2015.

[29] K. Zhang, M. Sadoghi, V. Muthusamy, and H.-A. Jacobsen, "Distributed Ranked Data Dissemination in Social Networks," in *ICDCS*, 2013.

[30] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram, "Scalable ranked publish/subscribe," *VLDB*, 2008.

[31] A. Carzaniga and A. L. Wolf, "Forwarding in a content-based network," in *Proceedings of ACM SIGCOMM*, 2003.

[32] M. Chen, S. Hu, V. Muthusamy, and H.-A. Jacobsen, "Congestion avoidance with incremental filter aggregation in content-based routing networks," Tech. Rep., 2011. [Online]. Available: http://msrg.org/papers/unsub-tr2011

[33] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th ed. USA: Addison-Wesley Publishing Company, 2009.

Appendix

This section presents the proofs for the properties used in the paper.

A. *Proofs of nature of subscription propagation*

We describe certain properties of how subscriptions are propagated in traditional acyclic content-based routing overlay networks.

In the following discussion, the *publisher host broker* (PHB) refers to the broker to which a publisher connects. Similarly, the *subscriber host broker* (SHB) is the broker to which a subscriber connects. Note that these are only logical designations, and any given broker may play the role of both a PHB and SHB.

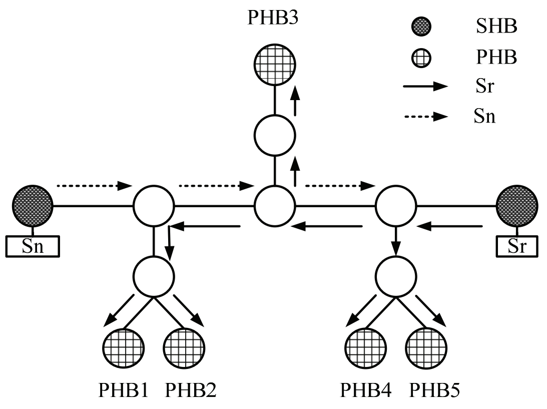


Fig. 11. Example sub. propagation

Consider two types of subscriptions: *root* subscriptions which are not covered by any other subscription in the system and *non-root* subscriptions which are covered by some subscription in the system. It turns out that the propagation paths of each class of subscriptions in an acyclic topology follows following properties.

Property 1. *Without the covering optimization, the propagation of any subscription S is a tree.*

Proof: Based on the subscription routing protocols [18], S is disseminated from the subscriber to the host brokers of publishers with intersecting advertisements. In this way a path is constructed between each intersecting publisher's PHB and the subscriber's SHB as required for reverse path forwarding. The dissemination of S is a tree rooted at the SHB of S with leaves at the potentially interesting publishers' PHB. ■

For the next property, let S_r and S_n be a root and non-root subscription, respectively, such that S_r covers S_n: $S_r \supset S_n$. Also, let $\{A_{S_r}\}$ and $\{A_{S_n}\}$ denote the set of advertisements that match S_r and S_n, and let T_{S_r} and T_{S_n} denote the subscription propagation trees of S_r and S_n.

Property 2. *With the covering optimization, T_{S_r} remains the same, and T_{S_n} can be reduced to a linear path.*

Proof: First, we note that the propagation of a root subscription S_r is not quenched by any other subscription since nothing else covers it, so its propagation is unaffected. It remains to show that the propagation of a non-root subscription S_n is a path. The argument proceeds as follows:

(1) From the definition of subscription covering, the advertisements that match S_n also match S_r: $\{A_{S_r}\} \supset \{A_{S_n}\}$.

(2) Then, based on the content-based routing algorithms, for each advertisement $A_i \in \{A_{S_n}\}$, there exists a path from the SHB of S_r to the PHB of A_i in T_{S_r}, and also a path from the SHB of S_n to the PHB of A_i in T_{S_n}. Since these two paths terminate at the PHB of A_i, they must intersect at some broker O_i. Moreover, the paths are identical from O_i to A_i.

(3) Every O_i must exist in the path from the SHB of S_r to the SHB of S_n. If this were not the case, then three paths would exist: SHB of S_r to O_i, SHB of S_n to O_i, and SHB of S_r to SHB of S_n. These paths would form a cycle which is impossible in an acyclic graph.

So, for any given advertisement A_i, S_n only needs to be propagated to O_i. Then, S_r can help S_n to show S_n's interest in the identical path. Since O_i must exist on the path between SHB of S_r and SHB of S_n. we can conclude that if a covering subscription S_r exists, then S_n is only propagated along a portion of the path between the SHBs of S_n and SHB of S_r. ∎

To illustrate the above property, consider the example in Fig. 11, where subscription S_r matches advertisements A_1-A_5, and so the propagation tree of S_r is rooted at broker A with leaves at brokers I, J, K, L, and M. S_n is covered by S_r, and S_n only matches $A1$, $A3$, and $A5$. From the figure we can see that the PHB of A_1 is broker I, and broker B is the intersection of paths A-I and E-I, so O_1 is B, and similarly, O_3 of advertisement A_3 is C, O_5 of advertisement A_5 is D, so the subscription S_n only needs to be propagated from broker A to broker D.

There are three cases that arise from Property 2 when we have a subscription S_n covered by S_r: (1) The furthest O_i is S_n. In this case, the subscription path of S_n is the single node consisting of the SHB of S_n. (2) The furthest O_i is S_r. Here the subscription path of S_n is from S_n to S_r. (3) The furthest O_i is some broker between S_r and S_n. The subscription path of S_n is then from S_n to the furthest O_i.

Intuitively, a non-root subscription S_n only needs to propagate far enough to "graft" onto an existing dissemination tree constructed by a covering subscription S_r, thereby avoiding the remaining propagation cost.

B. Non-decreasing similarity

Let S be a root subscription, and $NRS(S)$ the set of subscriptions whose propagation is quenched by the presence of S. The following property establishes how the similarity, ϕ, between S and $NRS(S)$, varies at each broker along the propagation path of S.

Property 3. *For root subscription S and a matched advertisement A, the similarity, ϕ, is non-decreasing at each broker on the path from the SHB of S to the PHB of A.*

Proof: Let $P(\cdot)$ denote the publications published by A, then $P^*(S) = P(NRS(S)) \cup (P(S) \cap P(INC(S)))$.

Each broker B along the path from the SHB of S to the PHB of A may lie on the propagation path or tree of a subscription S', which matches A and belongs to $NRS(S)$ or $INC(S)$. Furthermore, once S' is propagated to B, it is propagated to every subsequent broker on the path to the PHB of A. This is because in an acyclic overlay, there is only one path over which S and S' can traverse from B to the PHB of A.

Therefore, the subscription sets $NRS(S)$ or $INC(S)$ at a broker in the path is always a superset of those at the previous broker. This directly implies that $P(NRS(S))$ and $P(INC(S))$ are both non-decreasing, as $P(S)$ is fixed, we draw the conclusion that for a matching advertisement A, ϕ is non-decreasing. ∎