Middleware Systems Research Group (msrg.org) Technical Report, University of Toronto, November 2010

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

Mingwen Chen, Songlin Hu, Vinod Muthusamy, Hans-Arno Jacobsen, and Zhiyong Liu

***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

The publish/subscribe paradigm provides a powerful interaction model for a wide variety of applications including distributed workflow management [1], [2], [3], [4], algorithmic trading in financial markets [5], [6], online games [7], [8], network management [9], intrusion detection [10], and RSS filtering [11].

In many of these systems, filter aggregation is a common technique used by the pub/sub routers to reduce routing table size and improve routing performance. Such filter aggregations are an especially popular optimization in distributed content-based pub/sub systems, where nodes are addressed by a filter, or subscription, which describes the interests of the node. In Internet-scale pub/sub deployments, many users may share similar interests [12]. For example, in a stock trading application, many traders may be interested in trades concerning the same stock, and during a sporting event, many millions of fans may wish to receive updates on high-profile matches, teams, or players. In such cases, constructing routing paths for each subscriber independently may be prohibitively costly, but by exploiting the similarity among users' interests, the routing state can be significantly reduced.

One common technique to exploit similarity among subscriptions in content-based pub/sub systems is *subscription covering*, which takes advantage of the property that a router need not index a specific subscription when there is a more general one present [13], [14]. In a sense, a general subscription aggregates the interests of the more specific ones it *covers*. The covering optimization has many benefits including smaller routing tables, fewer subscription propagations, and faster routing decisions [15], [14].

While algorithms to maintain the correctness of content-based routing tables when using covering optimizations have been around for many years [13], what is not well known is that these techniques have a drawback, which is a problem confronted in previous work [16]. Consider a content-based router with a general subscription S and a set of more specific subscriptions $\mathbb{S}$ that are covered by S. The covering optimization will only forward S and avoid forwarding the subscriptions in $\mathbb{S}$. However, when S is to be removed, perhaps because the client is not interested in S any more, the subscriptions in $\mathbb{S}$ are no longer covered by any subscription, and it becomes necessary to forward all the subscriptions in $\mathbb{S}$. To maintain correctness of the routing tables, all the subscriptions in $\mathbb{S}$ are forwarded immediately before removing S. Since the set $\mathbb{S}$ can be large, this operation can impose severe congestion in the network, overwhelming the brokers that must process this burst of subscription traffic, and destabilizing 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.[1] In this case the system only stabilizes after several hours.

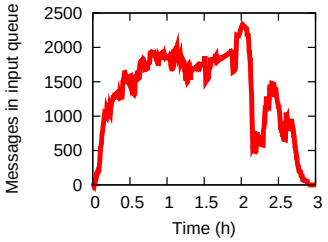


Fig. 1. Subscription bursts triggered by unsubscriptions can result in severe congestion that persists for a long time.

Many application scenarios are vulnerable to subscription bursts triggered by the removal of certain general subscriptions. For example, in a distributed workflow management system, each workflow activity would issue subscriptions that reflect its narrow interest in the events that trigger it [1], [2], [3]. 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 that were heretofore covered by S. Another application is a distributed gaming platform where players subscribe to events within a region of

[1]The methodology and configuration details of this experiment are presented in Sec. V.

the game world [7], [8]. Players may have different visibility in the game, perhaps due to the terrain and obstacles in the world, or the capabilities of the player's avatar. During the course of the game, as players' visibilities change, there will inevitably be moments where a subscription that covers many others needs to be removed or modified, resulting again in a subscription burst. A slightly more concrete scenario presents itself in the Mammoth distributed gaming platform [7]. In Mammoth, in addition to players subscribing to nearby regions, interest management servers subscribe to events in the entire gaming world, and these subscriptions therefore cover an incredibly large number of subscriptions. When an interest management server migrates, such as when a failure necessitates the election of a new server, there can be a subscription burst that destabilizes the system.[2] As another example, consider applications which exhibit disconnected operation [17]. For example, in a delivery agency, regional managers may subscribe to updates from delivery vans within their region, whereas a chief operating officer (COO) would be interested in the status of the entire fleet. As the COO travels during the day, her subscription, which covers those of the regional managers, needs to be removed and reissued at different locations. Each movement then may trigger messages that congest the system. The problem is exacerbated if the COO's subscription is continuously reissued at the nearest broker as she moves, as in mobile handoffs in cellular networks. Finally, this problem may occur in any news dissemination application, such as those propagating stock feeds to market analysts or delivering sporting event updates to interested fans [5], [11]. It is natural for users in these scenarios to express both broad and narrow interests, and as the set of subscriptions change there is a risk of subscription bursts. Essentially, the problem addressed in this paper may arise in any scenario where subscriptions are added and removed over time, and where there is a high degree of subscription covering.

The goal of this paper is to develop mechanisms to avoid congestion-induced instability of the content-based routing protocols while preserving the benefits of subscription aggregation techniques such as the covering optimization. There is an inherent tradeoff that must be made. For example, in the above scenario, 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 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 are 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 a subscription S depends on the similarity between S and the subscriptions that would replace it. If the similarity is high, there will be few wasted false positive messages, and it is worthwhile to preserve S.

This paper makes the following contributions. In Sec. III, certain properties of subscription propagation in content-based routing networks are formalized and proven. As well, a metric to capture the similarity between subscriptions is developed. The perfect merging of subscriptions is NP-hard [19], therefore the history of messages that match a given subscription is used instead to statistically approximate similarity. The similarity metric and subscription propagation properties above are then used in Sec. IV to develop a distributed algorithm to selectively 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. The results show that the algorithms presented in this paper provide significant benefits by avoiding severe network congestion that persist for several hours, reducing routing table sizes by about 50%, and decreasing message propagation delays by several orders of magnitude.

II. Related work

Content-based publish/subscribe routing: Content-based routing systems [14] have been developed for a variety of environments including dedicated acyclic overlays [20], distributed hash tables [21], [22] and wireless ad-hoc networks [23]. In this paper we focus on systems such as Siena [20] and 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 the 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 [14], and has been implemented in many pub/sub 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

[2]We make no claims on the performance of the Mammoth system, and only use its architecture here to motivate a potential problem. In particular, this paper addresses congestion in a content-based pub/sub system, a problem that does not necessarily manifest itself under the same conditions in the less expressive topic-based model employed in Mammoth.

remove redundant subscriptions from the network, maintain compact routing tables and reduce network traffic congestion.

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 [14], [13]. 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 [16] 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 [14].

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, 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. [19] 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 pub/sub: Congestion control in pub/sub systems differs from traditional congestion control found in other networking systems. Pietzuch *et al.* [25] presented a congestion control scheme based on the pub/sub messaging model, which was a combination of two congestion control mechanisms, the subscriber host broker-driven protocol and publisher host broker-driven protocol. The first mechanism managed the rate at which a subscriber host broker (SHB) requested missing data by sending NACKs upstream. The SHB maintained a NACK window to decide which parts of the message stream should be requested. Then, the NACK window was opened and closed additively depending on the level of congestion in the broker network. The change in recovery rate throughput was used for detecting congestion. The second mechanism regulated the rate at which new messages were published by a client and implement a scalable and low latency feedback loop between a publisher hosting broker and all subscriber hosting brokers. This could effectively adjust the rate of publishing new messages, allow brokers under recovery to eventually catch up, and other brokers to keep up. Although this paper successfully handled the problem of congestion caused by publication, it does not address the problem of congestion triggered by unsubscriptions, as described above.

III. Subscription propagation properties

We first analyze the basic issues related to subscription propagation in an acyclic content-based routing overlay, and then we identify and define a measure of similarity between a subscription that is to be removed and the set of subscriptions that must be propagated to replace it. These properties are fundamental to the subscription pruning algorithm developed in Sec. IV.

A. Nature of subscription propagation

In this section, we notice 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.

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 certain 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. ■

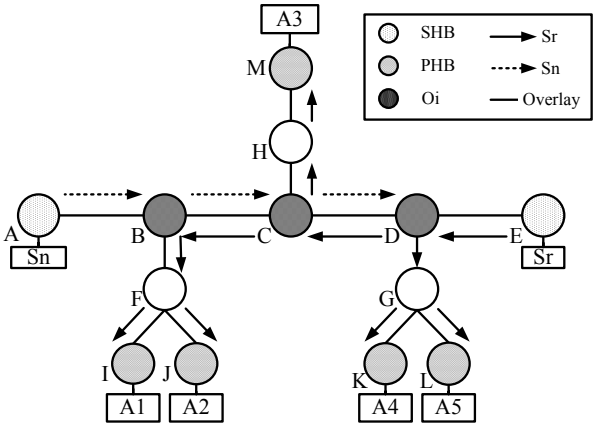


Fig. 2. Example subscription propagation

To illustrate the above property, consider the example in Fig. 2, where subscription S_r matches advertisements A_1-A_5, and so T_{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:

Case 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.

Case 2: The furthest O_i is S_r. Here the subscription path of S_n is from S_n to S_r.

Case 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. Subscription similarity

When a client removes a root subscription S, the subscriptions quenched by S will be activated. We denote these subscriptions as $NRS(S)$, the new root subscriptions of S. By Property 2, the propagation of subscriptions in $NRS(S)$ formed a path before S was removed, but may become a tree after S is removed. Since there may be many subscriptions in $NRS(S)$, and the size of their propagation trees may be large, the operation triggered by the removal of S can be very expensive and cause congestion.

However, if we can develop a metric to quantify the similarity between S and $NRS(S)$, then when the filter functions of S and $NRS(S)$ are in proximity, some portions of the subscription propagation tree T_S can be selectively preserved as an imprecise aggregation of $NRS(S)$. In other words, at specific brokers, there may be little benefit in removing the old filter of S and propagating new filters for $NRS(S)$.

It is difficult to quantify similarity using only the subscription filters. First, determining the filter similarity between S and $NRS(S)$ involves computing a merger of $NRS(S)$, which is NP-hard [19].

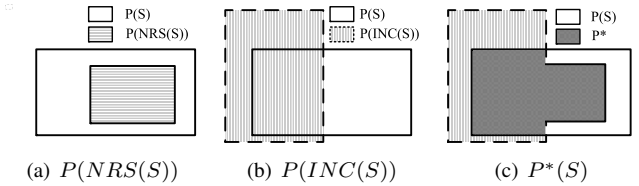
Moreover, the subscriptions may not be representative of the distribution of publications received by the subscribers. Suppose a root subscription S_1 indicates an interest for values in the range $[0, 100]$, and a covered subscription S_2 has an interest in values $[0, 50]$. Based only on these filters, 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 the range of interest of S_3 intersects that of S_1 but S_3 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.

Since we desire a measure of the *effective* filtering of subscriptions, in this section we develop a metric that calculates similarity based on the actual history of publications that match the subscriptions.

First, we define some notation that will be used. For a given subscription S, let $INC(S)$ refer to those subscriptions that intersect S but do not cover it or are covered by it, let $P(\cdot)$ denote the set of publications matched by one or

more subscriptions, and let $P^*(S) = P(NRS(S)) \cup (P(S) \cap P(INC(S)))$.



(a) $P(NRS(S))$ (b) $P(INC(S))$ (c) $P^*(S)$

Fig. 3. Relationships among $P(S)$, $P(NRS(S))$, $P(INC(S))$, and $P^*(S)$

$P^*(S)$ are the publications either matching $NRS(S)$ or $INC(S)$, after removal of S. From Fig. 3, we see $P^*(S)$ is a subset of $P(S)$. The similarity between S and $NRS(S)$ is then computed as $\phi = \frac{sizeof(P^*(S))}{sizeof(P(S))}$, where the value of ϕ is in the range $[0, 1]$.

C. Non-decreasing similarity

Note that the similarity, ϕ, between S and $NRS(S)$ differs at every broker, and it is too expensive to record the history of publications and compute the similarity at each broker. In this section, we develop an important property. By this property, we only need to record the history of publications at SHB of S, and other brokers can make use of the result computed at SHB of S.

Recall that for a given publisher, a root subscription S that matches the an advertisement A must be propagated along a path from the SHB of S to the PHB of A. It turns out that along this path, the similarity metric defined in Sec. III-B is non-decreasing. This property which is more formally defined below, is exploited in Sec. IV to selectively prune portions of a subscription's propagation tree.

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 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. ■

IV. Selective subscription tree pruning

As pointed out in Sec. III, the similarity between S and $NRS(S)$ increases along path of T_S. For every advertiser A, there exists a *critical broker* between the SHB of S and the PHB of A, where the similarity ϕ is large enough that it is more beneficial to replace S with $NRS(S)$ before this broker, and preserve S after this broker. Our pruning method seeks to determine the critical brokers in a distributed manner with little overhead. Since our algorithm is based on the actual history of publications, every SHB of subscription S records information about the publications matched by S. A point worth emphasizing is that only the SHB of S needs to maintain statistics on a window of publications that match S. Notably, a broker does not record the publications that only traverse through it. The statistics at the SHB are used by a lightweight and fully distributed algorithm to selectively prune portions of a subscription's propagation tree.

A. Statistics collection

As we mentioned above, only the SHB of S needs to record some statistics on a window of publications for S. As the similarity differs in every broker, two values are added to the header of publications, so as to collect the necessary statistics. The first one is *distance*, which records the number of overlay hops between the SHB of a subscription S and the nearest broker with another subscription S' that also matches the publication. The second one is *count*, which records the number of subscriptions interested in the publication as it propagates.

At every PHB, the broker initializes the *distance* and *count* values of a publication to zero. At every other broker that forwards the publication, if the publication matches more than one subscription, *count* is incremented by one and *distance* reset to zero. Otherwise, *count* is left unchanged, and *distance* is incremented by one.

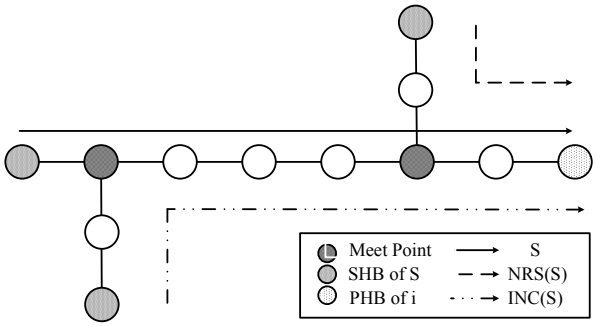


Fig. 4. Example publication propagation

For the scenario in Fig. 4, Broker A receives a publication from a publisher, and inserts the initial *distance* and *count* fields to the publication's header. At broker B *distance* is incremented to one, and *count* remains zero. At Broker C, the publication matches multiple subscriptions and so *distance* is reset to zero, and *count* incremented to one before forwarding to Brokers D and J. This process is repeated, and when the publication reaches Broker K, *distance* is two, and *count*

is one. At Broker G, again the publication matches two subscriptions, and so *distance* is reset to zero, and *count* becomes two. When the publication finally arrives at Broker I, both *distance* and *count* are two. From this, Broker I deduces that the nearest broker with a subscription that is also interested in the publication is two hops away.

Each broker that is the SHB for a subscription that matches a publication records the following information: *pubID*, the publication identifier; *advID*, the advertisement identifier; *subID*, the subscription identifier; and the *distance* and *count* values in the publication. For example, Table I shows what the database at Broker I might look like. The entries in the database can expire after a configurable time so that it maintains information about a sliding window of matching publications. Furthermore, these entries are purged when the associated subscriptions are unsubscribed.

pubID	advID	subID	distance	count
N1	A1	S1	6	1
N2	A1	S1	2	2
N3	A1	S1	2	2
N4	A1	S1	6	1
N5	A1	S1	7	1

TABLE I
PORTION OF THE DATABASE AT BROKER I IN FIG. 4

B. Subscription tree pruning

The *count* value represents the number of subscriptions interested in a given publication, and since it is the sum of the sizes of $NRS(S)$ and $INC(S)$, it can be used to estimate the size of $NRS(S)$. When a SHB receives an unsubscription for S, the maximum *count* value for S is fetched from the database. A small value suggests that no congestion will be caused by this unsubscribe operation and the unsubscribe operation proceeds in the traditional manner; otherwise, a selective pruning method is needed as described below.

Based on a given threshold of similarity, for each advertiser, the SHB can efficiently and locally determine the distance to the critical broker using the $distance$ value. For example if the threshold of similarity is 0.85, then for advertisement A, if the $distance$ value of more than 85 percent of the publications from A is less than 7, then the critical broker of A is 7 hops away in the path from SHB to the PHB of A. Then, a list of tuples is generated for the unsubscription message with each tuple containing 2 values: the identifiers of matched advertisers (*advID*), and the pruning distance from the SHB to the critical broker (*pruneDis*). These tuples are added to the header of the unsubscription message, and our pruning algorithm is performed with the propagation of the unsubscription message.

Each broker uses Algorithm 1 to determine whether to continue forwarding the unsubscription and prune the next hop of the subscription tree. For every neighbor U of the current broker, if all the advertisements from U have reached its critical broker, then it is unnecessary to delivery the unsubscription message to U. In other words, the remainder of the subscription tree T_S is preserved.

Algorithm 1: Handler of *Unsubscription Message*

```
Input: us ← an unsubscription message
forall the neighbor U of current broker do
    generate a new tuplelist newlist;
    newlist ← φ;
    forall the tuples {advID, pruneDis} ∈ unsub.tuplelist do
        if advID.sender = U and pruneDis > 0 then
            newlist ← newlist ∪ {advID, pruneDis − 1};
        end
    end
    if newlist ≠ φ then
        nus ← us;
        nus.tuplelist ← newlist;
        sent nus to U;
    end
end
```

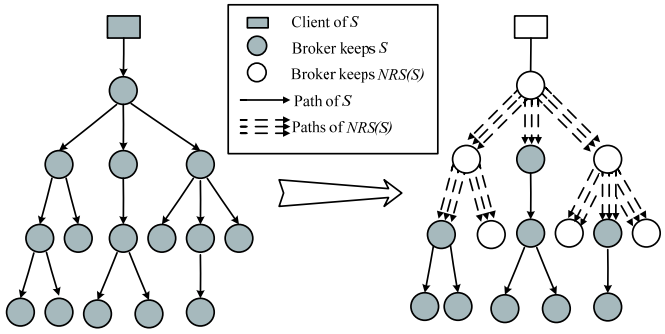


Fig. 5. A subscription tree before and after selective pruning.

Fig. 5 depicts an example of selectively pruning a subscription propagation tree T_S. The white broker indicate where S has been removed and replaced with subscriptions in $NRS(S)$, the dark brokers are where S has been preserved.

V. EVALUATION

In this section, we experimentally compare our selective subscription aggregation and pruning algorithm with traditional covering optimizations. The objective is to evaluate the performance of our algorithm under a variety of workloads and system load characteristics.

A. Methodology

1) Default setup: The protocols described in this paper have been implemented in the Java-based PADRES content-based pub/sub prototype.[3] 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.

The network topology in the experiments consists of 55 brokers, as shown in Fig. 6, with 23 inner broker nodes

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

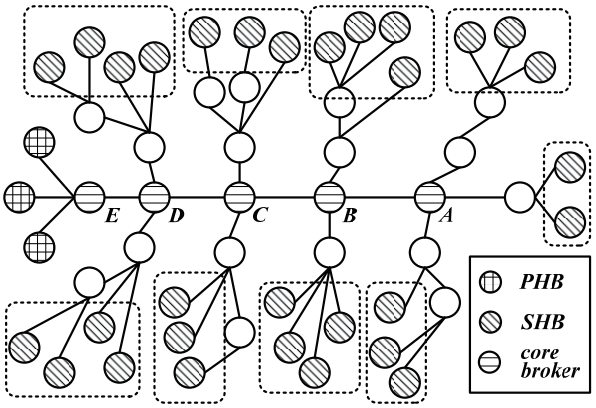


Fig. 6. Default broker topology

which do not host any publishers or subscribers, and 32 edge brokers. Of the inner brokers, the 5 brokers labeled A–E are denoted as core brokers. Publishers connect to the 3 edge brokers connected to broker E, and subscribers are randomly distributed among the remaining edge brokers. This might represent a messaging platform for the business workflows in a large enterprise with multiple departments and partners. The components in each workflow would subscribe to triggers of interest, and these workflows are invoked by publications from external clients [1], [2], [3].

Unless otherwise stated, about 30 000 subscriptions are issued at the beginning of the experiment from the subscriber hosting brokers with an interval of 1500 ms between subscriptions. This phase might correspond to the deployment of the workflow components.

During the course of the experiments, slightly over 21 000 publications are issued from the publisher hosting brokers at a rate of one every 400 ms.

The degree of covering among the subscriptions is quantified by the *covering degree* which is the average number of new root subscriptions triggered by an unsubscribe operation. Let R_i denote the number of subscriptions in the set $NRS(S_i)$, and N represent the number of root subscriptions. Given a set of subscriptions, the covering degree is then defined as $CovDeg = \frac{\sum_{i=1}^{N} R_i}{N}$. The default subscription covering degree in the experiments is 30.

The default similarity threshold for the selective pruning algorithm is set at 90%, which implies a 10% publication false positive rate.

2) Algorithms: The experiments compare the selective pruning algorithm proposed in this paper with three other algorithms described below.

Active covering: Active covering is 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: In the lazy covering algorithm, when a new subscription S covers existing uncovered subscriptions $\mathbb{S}$, the broker simply forwards S, and when S needs to unsubscribed, only subscriptions after S may need to be forwarded. While this algorithm forwards fewer subscriptions, over time it results in larger routing tables and hence slower publication matching and delivery performance.

Subscription packing: Subscription packing is an optimization of the active covering algorithm: when a set of subscriptions or unsubscriptions $\mathbb{S}$ need to be forwarded at once, they are forwarded in one message. While this algorithm reduces the number of messages sent, each subscription or unsubscription still needs to be processed individually by the recipient brokers.

3) Metrics: The experiments measure a number of metrics that are useful in understanding how the algorithms perform.

Publication propagation delay: This is the end to end latency of publications as they traverse from a publisher to a subscriber.

Input queue size: This is 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 a set of unsubscribe operations.

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

Triggered NRS messages: This metric measures the number of hops traversed by the NRS subscription messages that are triggered by an unsubscribe operation.

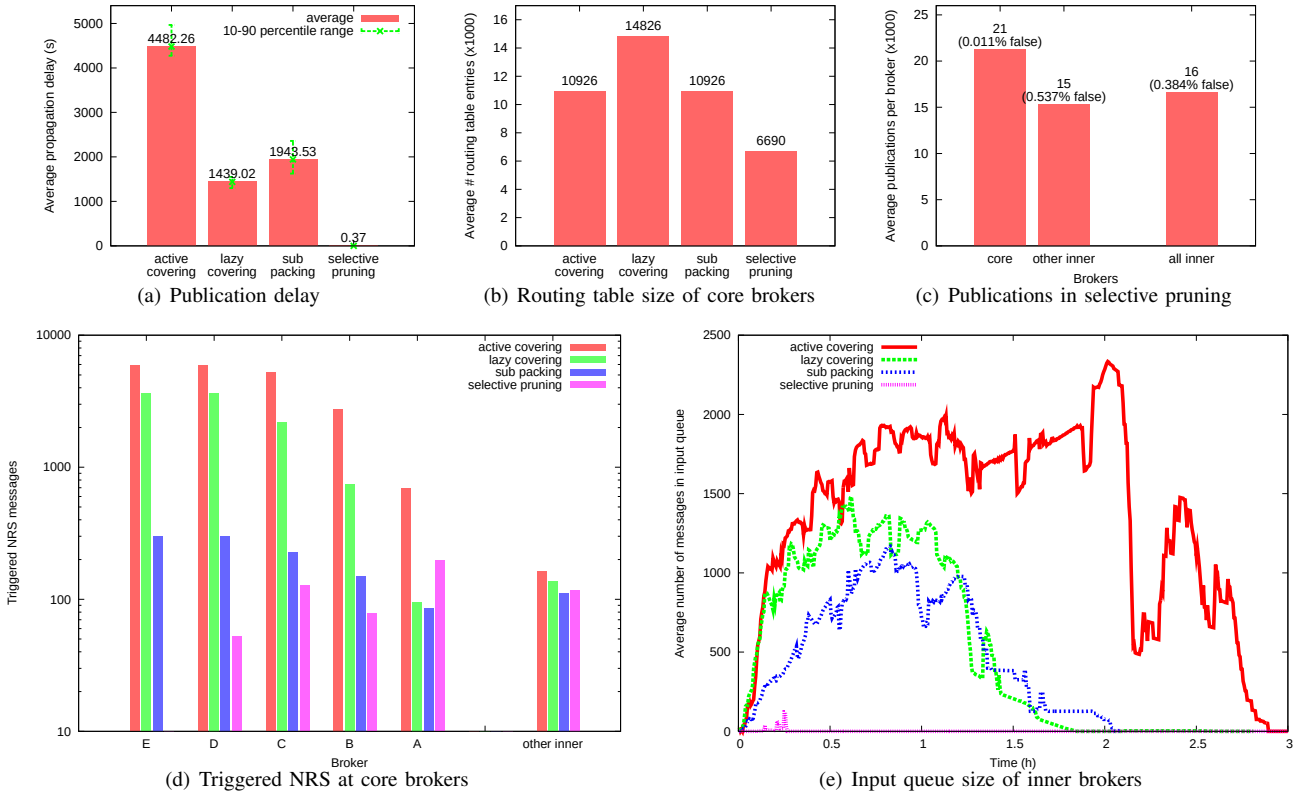
False positive publications: This is a count of the number of publications that are unnecessarily forwarded towards uninterested subscribers. Only the selective pruning algorithm has false positives.

B. Comparisons among algorithms

The first set of experiments compare the selective pruning algorithm proposed in this paper with the traditional active covering algorithm, as well as the lazy covering, and packing algorithms.

In this experiment 300 of the 30 000 subscriptions in the system are unsubscribed at an interval of 500 ms between unsubscriptions. While this means all the unsubscriptions are issued within 150 s, the experiment is allowed to run for several hours to measure how the algorithms process any resulting message bursts and congestion.

First, we examine the propagation delay of publications. While publications continue throughout the experiment, the delay results only consider a set of 300 publications matching subscribes at all 32 edge brokers that are issued 600 s after the unsubscribe operations. Fig. 7(a) plots the average propagation delay for the four algorithms, along with 10 and 90 percentile values in the distribution of delays. The results show that the average publication delay across the experiment



(a) Publication delay (b) Routing table size of core brokers (c) Publications in selective pruning

(d) Triggered NRS at core brokers (e) Input queue size of inner brokers

Fig. 7. Comparisons of active covering, lazy covering, subscription packing and selective pruning algorithms

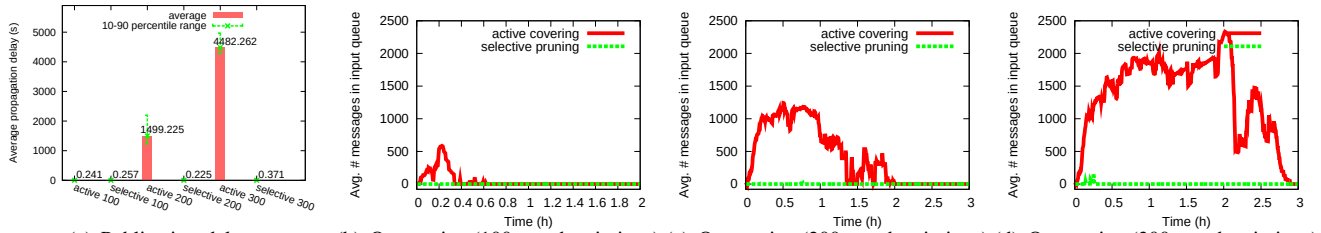
with the active covering algorithm is over 4400 s, while the lazy covering and subscription packing algorithms deliver publications in less than half this time. The selective pruning algorithm, however, performs substantially better, with average delays of only about 0.4 s, an improvement of more than 99%. Moreover, congestion subsides quickly with the selective pruning algorithm, and the average steady state publication delay falls to 200 ms. Next, we will investigate the causes behind these publication delays.

Fig. 7(d) plots (on a log scale) the number of triggered NRS messages at the core brokers A–E, and shows that with active and lazy covering, the number of NRS messages 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 5900 messages at broker E. Similarly, lazy covering imposes almost 100 messages at broker A and over 3600 at E. Subscription packing, by combining the NRS messages associated with an unsubscribe operation, reduces this to fewer than 300 messages. The selective pruning algorithm further improves on this performance with only about 200 messages at broker A, decreasing to 0 NRS messages at broker E. Notice that the selective pruning algorithm triggers fewer messages near the publisher, which is the opposite behaviour of the algorithms. Fig. 7(d) also confirms the claim made in Sec. III-C that the selective pruning algorithm is more inclined to preserve subscription trees at brokers closer to a publisher. Finally, the results show that number of triggered NRS messages at the remaining inner brokers is much less than that at the core brokers, and there is no significant difference among the four algorithms.

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 the unsubscriptions. As shown in Fig. 7(e), the queue size of core brokers explodes to almost 2000 messsages with the active covering algorithm. The queue size in the lazy covering is smaller but still experiences a large burst of up to 1500 messages, and the maximum queues with the packing algorithm reach almost 1200 messages. However, the selective pruning algorithm presented in this paper far outperforms all of these with only a slight increase in the input queue size. More importantly, the selective algorithm stabilizes very quickly, while the active and lazy covering only manage to slightly reduce the queue size even after one hour into the experiments. These large long-lived queues indicate that the system is overloaded and cannot function normally.

The degree of congestion in Fig. 7(e) may seem surprising, but additional profiling revealed that during periods of congestion the brokers processed publications and subscriptions at a rate of only 3.3 per second. Futhermore, the large publication delays in Fig. 7(a) are consistent with queuing theory results which show that queuing delays increase rapidly as the message arrival rates approach the link capacity [26].

It should also be noted that the burst of subscriptions in $NRS(S)$ triggered by an unsubscription of S do not only occur when the unsubscription is first issued. Instead, as the



(a) Publication delay (b) Queue size (100 unsubscriptions) (c) Queue size (200 unsubscriptions) (d) Queue size (300 unsubscriptions)

Fig. 8. Varying number of unsubscriptions

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 $NRS(S)$ that exists at that broker. This explains why the queue lengths in Fig. 7(e) 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 can decrease the number of triggered NRS messages but does nothing to reduce the computation overhead for processing these messages since they must still be processed individually by their receiving brokers. Taken together, Figs. 7(d) and 7(e) demonstrate that although fewer than 300 messages are triggered by this algorithm, the network can also be severely congested with thousands of publications and subscriptions pending in the brokers' queues.

In addition to triggering fewer messages, the selective pruning algorithm also resulted in smaller routing table sizes in this experiment. Fig. 7(b) 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 the active covering algorithm achieves very compact routing tables (only 1000 routing table entries for 30 000 subscriptions). The lazy covering algorithm, on the other hand, relies on the order that subscriptions are issued and only opportunistically benefits from covering relationships and thus results in larger routing tables. However, the selective pruning 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 selective algorithm only manifest when covering subscriptions are unsubscribed, so it may not always outperform the active covering algorithm.

The selective pruning algorithm introduces two sources of overhead, but the results show the overhead to be minimal. First, Fig. 7(c) presents the average publication traffic among the core and inner brokers with the selective pruning algorithm. Of this publication traffic, only about 0.38% of it are false positives. As well, the false positive traffic is skewed towards the more lightly loaded nodes: Fig. 7(c) shows that the more heavily loaded core brokers only experience a 0.01% false positive rate.

The other overhead of the selective pruning 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 data was stored in an Apache Derby database, the largest of which contained about 14 000 entries requiring about 12 MB of storage. The average broker's database was an even more modest 4.5 MB. More importantly, the results presented above on a real running implementation of the protocol show that, in spite of any overhead, the selective pruning algorithm far outperforms the traditional ones.

C. Varying number of unsubscriptions

In this section, we investigate the effects of varying the number of unsubscriptions issued. Since the results in Sec. V-B showed that the performance of the active covering, lazy covering and packing algorithms are similar, here we compare the selective pruning algorithm with only the active covering algorithm—the most widely used one.

Fig. 8 shows the performance of the algorithms when the number of unsubscriptions issued over a 150 s interval varies from 100 to 300.

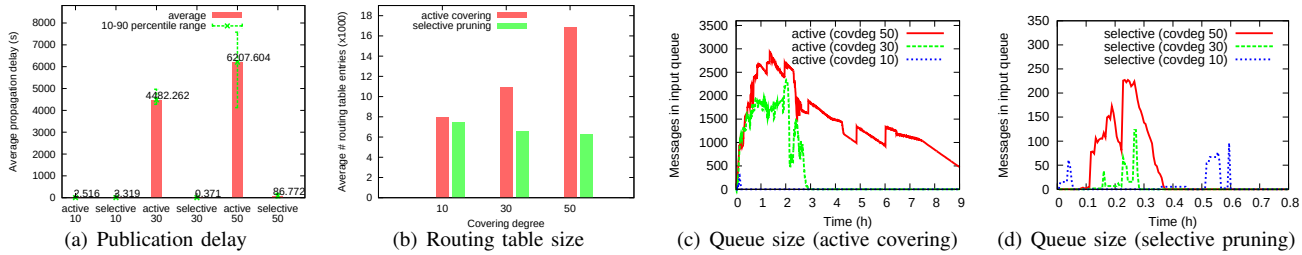
The publication delay results In Fig. 8(a) shows that both the active covering and selective pruning algorithms perform well, delivering publications in about 20 ms when there are only 100 unsubscriptions. However, the performance of the active covering algorithm quickly and substantially degrades to over 1200 s and 2000 s as the number of unsubscriptions increases to 100 and 300, respectively. Meanwhile, the selective pruning algorithm maintains the same publication delay performance.

The cause of the publication delays is the congestion arising from the subscription bursts in the active covering algorithm. Figs. 8(b), 8(c), and 8(d) show how the magnitude and duration of the input queues dramatically increase with the number of unsubscriptions under the active covering algorithm. The selective pruning algorithm, however, is much better at avoiding large congestion.

D. Varying subscription covering degree

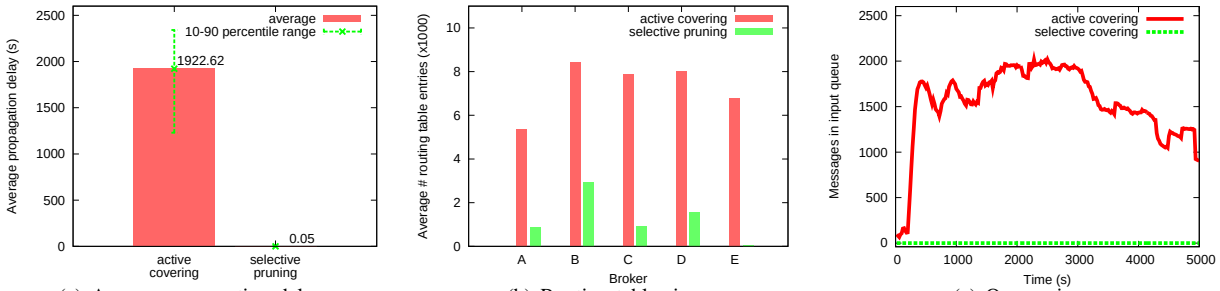
This section investigates the effects of varying the covering degree among the subscriptions in the workload from a low degree of covering of 10 to a high of 50.

Beginning again with the publication delays, Fig 9(a) shows that the active covering algorithm is sensitive to the degree of



(a) Publication delay (b) Routing table size (c) Queue size (active covering) (d) Queue size (selective pruning)

Fig. 9. Varying subscription covering degree



(a) Average propagation delay (b) Routing table size (c) Queue size

Fig. 10. Clustered workload

covering in the subscription population but the selective pruning algorithm provides relatively stable performance. Moreover, the results indicate a wide distribution in the publication delays under the active covering algorithm.

Fig. 9(b) shows that after unsubscribe operations, the routing table sizes sharply increase with increasing covering degree of subscription workloads for the active covering algorithm. More interestingly, since a larger covering degree gives more opportunities for filter aggregation, the selective pruning routing table sizes decrease with increasing covering degree of subscription workloads. All in all, the benefits of the selective algorithm accrue as the covering degree increases.

Figs. 9(c) and 9(d) 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 $NRS(S)$ sets, which are prone to lead to subscription bursts when the root subscription S is removed. For example, when the covering degree is 10, the active covering and selective pruning algorithm both result in relatively minor and short-lived congestion. When the covering degree increases to 30, the active covering algorithm, in Fig. 9(c), experiences bursty traffic with queue lengths of about 2300 messages. This trend continues with the active covering performing worse with larger covering degrees; when the covering degree is 50, the queue sizes continue to increase an hour into the experiment and remain even after 9 hours. Contrast this behavior to the selective pruning algorithm, in Fig. 9(d), which only experiences a short-lived congestion of less than 250 messages for all covering degrees. (Notice that both the time and queue size scales in Figs. 9(c) and 9(d) are different.)

E. Clustered workload

We now evaluate the algorithms using a workload where the subscriptions are clustered in the network [27]. In particular, subscriptions with covering relationships are distributed to the same cluster, with each cluster consisting of 2 to 4 adjacent brokers as depicted by the regions in Fig. 6. We revert to the default subscription covering degree of 30 for this experiment.

Fig. 10(a) shows the average propagation delay of 300 publications issued 600 s after the unsubscribe operations. We see that the selective pruning algorithm delivers publications in about 50 ms, which is less than 3% of the 2000 s average delays experienced when the active covering algorithm is used.

In terms of the routing table sizes in Fig. 10(b), the selective covering algorithm achieves substantially smaller tables. For example, among core brokers, the average table size with the selective pruning algorithm is about 1200 compared to about 7300 for the active covering algorithm, an impressive 82% reduction. What we see here is that clustering subscriptions makes it more likely that a root subscription can be left in place without suffering from too many false positives.

The input queue lengths are plotted in Fig. 10(c). We see that the active covering algorithm performs much worse when subscriptions are clustered since this tends to concentrate the effects of the congestion caused by the subscriptions in $NRS(S)$. Despite having issued all the unsubscriptions by 150 s into the experiment, the message bursts only begin to drop after 4000 s. The selective covering algorithm, however, has no congestion because the clustering of subscriptions in the topology offers more opportunities for filter aggregation, and almost all the triggered NRS are replaced by a broader subscription. As well, compared to Fig. 7(e), we observe that the benefits of the selective pruning optimization are

more evident when the subscriptions are clustered instead of scattered.

F. Mobile scenario

The following set of experiments consider a more dynamic scenario in which subscriptions are unsubscribed and resubscribed at different brokers. The experiment is designed to reflect scenarios such as a distributed gaming application.

The setup consists of a 5000 subscribers and 40 publishers distributed randomly across the brokers in the topology shown in Fig. 11. In a distributed online game, each broker may be responsible for managing a portion of the game world, so as players move in the game, they will reconnect to different brokers in the network [7], [8].

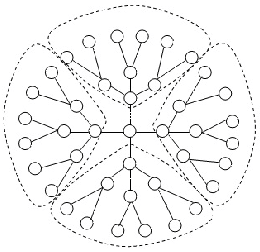
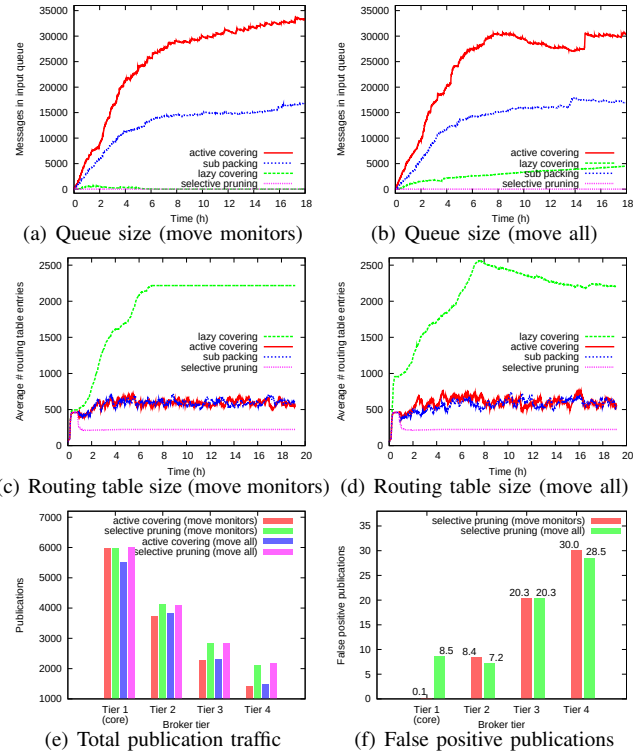


Fig. 11. Tiered topology for the mobile scenario



(a) Queue size (move monitors) (b) Queue size (move all)

(c) Routing table size (move monitors) (d) Routing table size (move all)

(e) Total publication traffic (f) False positive publications

Fig. 12. Mobile scenario

In addition, there are four subscriptions, one per region in Fig. 11 that are interested in all publications. We refer to these subscribers as *monitors* and they may represent interest management servers in an online game. As load conditions may trigger migrations of these interest management servers, these subscriptions may need to be resubscribed at other brokers.

We consider two cases: one where only the monitors move at a rate of one movement every 2 minutes for a total of 6 hours, and another where in addition to the movement of the monitor subscriptions, the others subscriptions move at a rate of 60 movements per minute for the entire duration of the experiment. The non-monitor subscriptions only move within their region in the topology. In both cases, publishers scattered among the brokers do not move.

Fig. 12(a) shows the congestion at the core broker when only the monitor subscriptions move. We see that the active covering algorithm suffers from a large and growing input queue, whereas the selective pruning algorithm has virtually no congestion. When all subscriptions move in Fig. 12(b), there is no significant change in the congestion, indicating that it is the movement of the broad monitor subscriptions that cause the congestion.

In terms of the routing tables, Fig. 12(c) shows that with the active covering algorithm the average number of entries in the broker routing tables fluctuates during the experiment. These fluctuations are a consequence of the active covering algorithm attempting to maintain compact routing tables when covering subscriptions are removed, particularly the monitoring subscription. This is why there the introduction of the movement of smaller subscriptions in Fig. 12(d) has little effect on the routing table size. The selective pruning algorithm, on the other hand, has a much smaller and more stable routing state in both cases. The reason for the stable performance is that the selective pruning algorithm effectively maintains the monitor subscriptions even after they move away, and so after a few movements, these subscriptions exist virtually everywhere in the network and "hide" the movement of the other subscriptions. Interestingly, this means that with subscription pruning, the presence of movement can actually result in smaller routing tables than when there is no movement.

Of course, preserving these subscriptions also increases the false positive publication traffic, but it turns out that this traffic is distributed mostly among the more lightly loaded brokers in the system. Fig. 12(e) shows the average number of publications traversing each tier of brokers in the topology. In both algorithms, the brokers in the upper tiers are more heavily loaded, with selective pruning imposing more traffic due to the false positives. However, when we isolate this false positive traffic in Fig. 12(f), we see that it is the more lightly loaded lower tier brokers that are required to carry most of the false positive load, helping to distribute publication load.

VI. Conclusion

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 selectively 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 selective 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 selective covering algorithm with the traditional ones reveal severe congestion under the traditional algorithms that are virtually eliminated with the selective 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 selective algorithm, on the other hand, hardly experienced any 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] G. Li, V. Muthusamy, and H.-A. Jacobsen, "A distributed service-oriented architecture for business process execution," *ACM Trans. Web*, vol. 4, no. 1, 2010.

[2] D. Wodtke, J. Weisenfels *et al.*, "The Mentor project: Steps toward enterprise-wide workflow management," in *ICDE*, 1996.

[3] P. Muth *et al.*, "From centralized workflow specification to distributed workflow execution," *JII*, vol. 10, no. 2, pp. 159–184, 1998.

[4] C. Schuler, H. Schuldt, and H.-J. Schek, "Supporting reliable transactional business processes by publish/subscribe techniques," in *TES*, 2001.

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

[6] P. Pietzuch, B. Shand, and J. Bacon, "Composite event detection as a generic middleware extension," *IEEE Network*, 2004.

[7] J. Kienzle, C. Verbrugge, K. Bettina, A. Denault, and M. Hawker, "Mammoth: a massively multiplayer game research framework," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, ser. FDG '09, 2009, pp. 308–315.

[8] P. B. Beskow, K.-H. Vik, P. Halvorsen, and C. Griwodz, "Latency reduction by dynamic core selection and partial migration of game state," in *NetGames*, 2008.

[9] T. Fawcett *et al.*, "Activity monitoring: Noticing interesting changes in behavior," in *SIGKDD*, 1999.

[10] B. Mukherjee *et al.*, "Network intrusion detection," *IEEE Network*, 1994.

[11] I. Rose, R. Murty *et al.*, "Cobra: Content-based filtering and aggregation of blogs and RSS feeds," in *NSDI*, 2007.

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

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

[14] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, Aug. 2001.

[15] 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 *International Conference on Distributed Computing Systems (ICDCS)*, Columbus, Ohio, 2005.

[16] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen, "Transactional mobility in distributed content-based publish/subscribe systems," in *Proceedings of the 2009 IEEE International Conference on Distributed Computing Systems (ICDCS 2009)*, Montreal, Canada, 2009.

[17] D. Conan, S. Chabridon, and G. Bernard, "Disconnected operations in mobile environments," in *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, ser. IPDPS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 118–.

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

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

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

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

[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 *Feature Interactions in Telecommunications and Software Systems VIII (ICFI 2005)*, Leicester, UK, 2005.

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

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

[27] A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and L. Zhang, "Clustering algorithms for content-based publication-subscription systems," in *ICDCS*, 2002.