Adaptive Multi-Path Publication Forwarding in the Publiy Distributed Publish/Subscribe System

Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen
{*reza, jacobsen*}*@eecg.utoronto.ca*

Middleware Systems Research Group (MSRG), University of Toronto

November 27, 2011
Toronto, Ontario, CANADA

Abstract

Fine-grained filtering capabilities of content-based Publish/Subscribe (P/S) overlays often lead to scenarios in which publications pass through brokers with no local matching subscribers. Processing of messages at these pure forwarding *brokers amounts to inefficient use of resources and should ideally be avoided. Accomplishing this goal requires a rethought of how the P/S overlays are maintained. In this paper, we develop an approach that largely mitigates this problem by building and adaptively maintaining a highly connected overlay mesh superimposed atop a low connectivity* primary *overlay network. While the primary overlay network provides basic forwarding routes, the mesh structure provides a rich set of alternative forwarding choices which can be used to bypass pure forwarding brokers. Moreover, the overlay mesh provides unique opportunities for load balancing and congestion avoidance. Through extensive experimental evaluation on a cluster and Planetlab, we compare the performance of our approach with that of conventional P/S algorithms as baseline. Our results indicate that our approach improves publication delivery delay and lowers network traffic while incurring negligible computational and bandwidth overhead. Furthermore, compared to the baseline, we observed significant gains of up to* 115% *in terms of system throughput.*

1 Introduction

Flexibility, scalability and loose coupling provided by the Publish/Subscribe (P/S) model has led to its adoption in a variety of enterprise, datacenter and wide-area network environments [14, 28, 11, 15, 17]. Microsoft, Google and Yahoo for instance use P/S service for end-user notification delivery [1], data dissemination in large-scale server farms [15], and in distributed data storage systems [11]. In enterprise settings, P/S has appeared in many contexts and standards including Enterprise Service Bus (ESB) [20, 17], algorithmic trading [25], WS-Notifications and WS-Eventing. In wide-area networks, P/S messaging has been used in push-based RSS feeds [24], global supply chain data exchange networks [14], and as a potential addressing and routing paradigm for the future Internet [23].

Wide-spread adoption of P/S further underlines the significance of scalable architectures that can efficiently utilize network bandwidth and achieve low message delivery delay. To this end, a distributed content-based P/S system deploys a set of dedicated application layer routers (*a.k.a.* brokers) to form an overlay network [13, 4, 22, 7, 12, 26, 9]. Clients connect to brokers and are offered the flexibility to specify fine-grained filtering constraints on the content of publications they would be interested to receive. Publications that satisfy these constraints are forwarded through the overlay towards all brokers where interested subscribers reside.

In a large network, it is infeasible to maintain full connectivity and it is imperative to only use a selective set of all links. Furthermore, the set of subscribers to which a given publication must be delivered is highly variable and cannot be determined in advance. This makes it *challenging to build optimal dissemination overlays that only span to brokers with interested local subscribers*. As a result, existing P/S systems use a shared dissemination overlay that may forward publications through uninterested (*a.k.a. pure forwarding*) brokers. Processing of publications at pure forwarding brokers increases message hop count and propagation latency, and amounts to inefficient use of bandwidth.

To improve the overlay and lower the number of pure forwarders, reconfiguration techniques incrementally modify the overlay by adding links between brokers with similar subscriber interests and removing links between those with less similarity [29, 3, 19]. Altering overlay links in this manner has a large network-wide footprint and while beneficial to some publication flows it can at the same time be detrimental to many others. Furthermore, each reconfiguration involves coordinated updates to routing tables of many brokers, a process that is slow and costly. Alternatively, clustering [10] techniques group subscribers with similar interests together and place them close to brokers where publishing sources reside. In a generic Internet-scale P/S system where clients have widely varying interests, the performance of such clustering schemes is not guaranteed. Also, a clients with multiple subscriptions may be prescribed to connect to different brokers, an inconvenience that should ideally be avoided.

Another common problem in existing P/S systems is that overlay forwarding paths are set up in a *fixed end-to-end manner*: *A publication is sent over a fixed path to a destination regardless of whether or not it matches subscribers at intermediate brokers along the path.* This approach inevitably results in a large number of pure forwarders. This deficiency could be mitigated if the overlay connectivity between sources and destinations offered a multitude of redundant forwarding paths giving brokers a chance to pick the best forwarding path based on the relative location of matching subscribers.

To this end, we propose an adaptive overlay management and dynamic routing approach that constructs a highly connected mesh structure atop a *primary overlay network*. The primary network offers basic connectivity among brokers and may use existing routing algorithms. By monitoring ongoing traffic in the primary network, brokers identify popular transit routes and establish additional communication links, referred to as *soft links*. Soft links construct a highly connected overlay mesh and provide a rich set of alternative paths between each source and

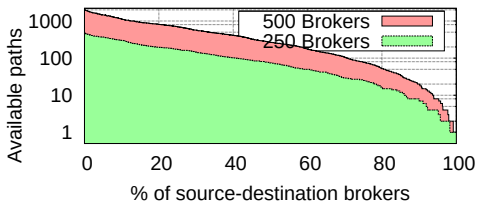


Figure 1: Diversity of available end-to-end forwarding paths in networks of 120 and 250 brokers (log scale).

destination broker. Figure 1 illustrates a snapshot view of the number of end-to-end paths in a running system using this scheme. About 40% of brokers in a network of size 120 have at least one hundred different forwarding paths among them. This is increased to more than $1,000$ for 13% of brokers in a network of size 250. All these routes are readily available for forwarding and determination of which path to take is based on the relative location of matching subscribers which takes place at runtime. This is in contrast to most P/S algorithms [7, 13, 12] which use *fixed end-to-end forwarding paths* and send a message to a destination broker over the same path regardless of whether or not it matches subscriptions at intermediate brokers. The premise of our approach is that availability of a very large set of alternative routes gives brokers the opportunity to consciously use the path that is best suitable for delivery of the message to all interested subscribers while incurring fewer pure forwarders.

To forward publications in the overlay mesh and determine the relative location of matching subscribers brokers rely on knowledge of their neighboring brokers within a certain distance, denoted by configuration parameter Δ. This knowledge enables a broker to anticipate how publications flow within its Δ-neighborhood and which alternative routes towards the destinations incur fewer pure forwarders. Furthermore, brokers monitor ongoing traffic to choose the best set of soft links based on three criteria: Avoiding pure forwarding brokers, avoiding slow primary links, and bypassing congested network hotspots. In our approach, maintaining the overlay mesh using soft links requires only local updates to routing tables. This is a light-weight process and a significant improvement over full overlay reconfiguration techniques [29, 3, 19] which require coordinated updates to several brokers' routing tables.

In this paper, we make the following contributions: *(i)* A scheme to adaptively maintain a highly connected mesh for P/S overlays; *(ii)* A scheme to update brokers' routing tables based on network connectivity with no need for coordination among neighbors; *(iii)* A number of forwarding strategies and efficient cache data structures to realize them; *(iv)* A traffic profiling technique to identify popular transit routes within the overlay; and *(v)* Comprehensive experimental evaluation using a Java-based prototype implementation, called *Publiy* (*Publiy* is an open-source project).

The rest of the paper is organized as follows: Sections 2 and 3 present how the brokers' overlay and subscription routing tables are maintained, respectively; Section 4 elaborates on our publication forwarding strategies; Section 5 presents our overlay management scheme; and Section 6 reports experimental evaluation results. We review the related work in Section 7 and conclude the paper in Section 8.

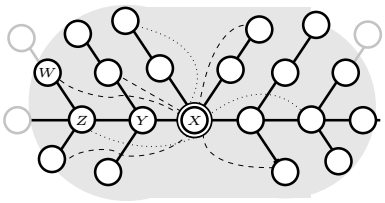
2 Overlay Maps

We assume that there is an initial P/S overlay that provides basic connectivity among brokers. We refer to this overlay and its links as the *primary network* and *primary links*, respectively. In this section, we elaborate on brokers' internal data structures, namely the *Master Overlay Map* and *Working Overlay Map*. Roughly speaking, the former reflects a partial view of the *primary network* which is relatively stable and changes infrequently. The latter data structure, however, acts as a cache and provides a dynamic view of a superimposed *overlay mesh* atop the primary network.

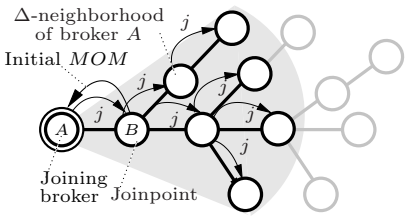
2.1 Master Overlay Map

Brokers store a *partial* view of the primary network in a local data structure called the *Master Overlay Map* (MOM). This partial view is in the form of a subgraph centered at the broker and contains all brokers and their primary links located within distance Δ (i.e., Δ-*neighborhood*). The shaded area in Figure 2(a) illustrates Broker A's Δ-neighborhood in a sample network where $\Delta = 3$.

A broker's first primary link is created when it contacts an existing system broker, called a *joinpoint*, and requests to join the system. For this purpose, the newcoming broker may rely on an external discovery service which considers access privileges and broker load conditions before determining the joinpoint. If the joinpoint accepts the request, it will reply with the portion of its own overlay map that falls within distance Δ of the joining broker. This information is used to initialize the MOM of the joining broker. Furthermore, the joinpoint notifies other existing brokers within distance Δ about the new broker so they also insert the joining broker and its primary link in their overlay maps. Figure 2(b) illustrated the join process for Broker A.



(a) Primary links, soft links, and candidate links are shown with solid, dashed and dotted lines respectively.



(b) Upon join, A receives its initial Master Overlay Map(MOM) from B, and Δ-neighborhood brokers update their MOMs.

Figure 2: Shaded area represents A's Δ-neighborhood ($\Delta = 3$).

2.2 Working Overlay Map

Other than primary links, brokers possess two other types of links, namely *soft links* and *candidate links*. These links construct an overlay mesh that is superimposed atop the primary network. Unlike primary links, soft and candidate links change frequently in order to enable quick adaptation to changes in network traffic and load conditions (details described later). In order to accommodate this level of dynamism and provide an efficient way to use the connectivity of the overlay mesh for publication forwarding, we devise the *Working Overlay Map* (WOM) data structure. WOM is derived from the MOM and transforms the broker's initial knowledge of the primary network into a concise representation in accordance to its current set of primary, soft and candidate links. WOM is *reconstructed locally upon every change to the broker's links* and acts as a pre-computed cache for efficient publication forwarding. In what follows, we describe the steps in construction of WOM from the perspective of Broker A shown in Figure 3.

Edge contraction: Initially, A's WOM contains all brokers and primary links in MOM, as well as A's own non-primary links. Broker A refines this graph by considering its neighboring brokers in descending order of distance in the primary network. For each such broker, v, with no direct link, the edge between v and a closer broker w is contracted and v is substituted with w by making all edges incident (connected) to v incident to w. Figures 3(a) and 3(b) illustrate A's MOM and WOM. At the end of this process, *WOM forms a graph that only contains neighboring brokers to whom A maintains a direct link.*

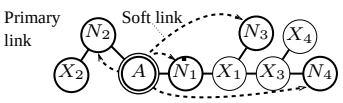
Broker array: Broker identifiers in the resulting graph are sorted in an array denoted by $BrokerArr_A$, in ascending order of their distance in WOM:

$$dist(BrokerArr_A[i]) \leq dist(BrokerArr_A[i+1]).$$

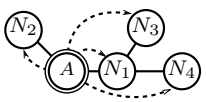
Henceforth, N_i, refers to the i^{th} broker in this array.

Auxiliary sets: For each Broker N_i, three auxiliary sets are computed that contain identifiers of neighbors located in different relative positions w.r.t. A and N_i (see Figure 3(c)). These auxiliary sets are as follows:

- $BetweenSet(A, N_i)$ contains N_i and brokers located on the path between A and N_i in the primary network. In Figure 3, $BetweenSet(A, N_4) = \{N_1, N_4\}$.



(a) MOM: A's primary network view



(b)WOM: A's mesh view

Brkr	Dist.	Substitutes	BetweenSet	BehindSet	BeyondSet
N_1	1	X_1, X_3, X_4	N_1	N_2	N_1, N_3, N_4
N_2	1	X_2	N_2	N_1, N_3, N_4	N_2
N_3	2	$\varnothing$	N_1, N_3	N_2	N_3
N_4	2	$\varnothing$	N_1, N_4	N_2	N_4

(c) Pre-computed information in A's WOM($\Delta = 4$)

Figure 3: Master and working overlay maps at Broker A.

- *BeyondSet*(A, N_i) contains brokers located downstream of N_i from A's point of view (including N_i). In Figure 3, *BeyondSet*$(A, N_1) = \{N_1, N_3, N_4\}$.
- *BehindSet*(A, N_i) contains brokers located downstream of A from N_i's point of view. In Figure 3, *BehindSet*$(A, N_1) = \{N_2\}$.

Figure 3(c) illustrates the auxiliary sets of Broker A.

We have now covered how brokers' overlay maps are updated. Next, we discuss routing table maintenance.

3 Subscription Routing Tables

Brokers maintain their routing tables in a similar manner as their views of the primary network and overlay mesh. More specifically, there are two routing tables, namely *Master Subscription Table* and *Working Subscription Table*: Subscription entries in the former table constructs end-to-end forwarding paths in the primary network. The entries in the latter, however, adapts these paths to the current state of the overlay mesh as reflected in the WOM.

3.1 Master Subscription Table

We introduce the notion of *subscription anchor* used to store subscription information in brokers' routing tables. From the perspective of a broker, an *anchor* for a subscription is a broker located up to Δ hops closer to the issuing subscriber (anchor of local subscribers points to the broker itself). Anchors are used to forward matching publications hop-by-hop (or multiple hops at a time) towards subscribers. The advantage of using anchors is that availability of neighborhood knowledge allows brokers to determine the relative location of all matching subscribers and choose the actual publication forwarding path from a wealth of alternative routes (strategies that are used for this purpose are discussed in Section 4).

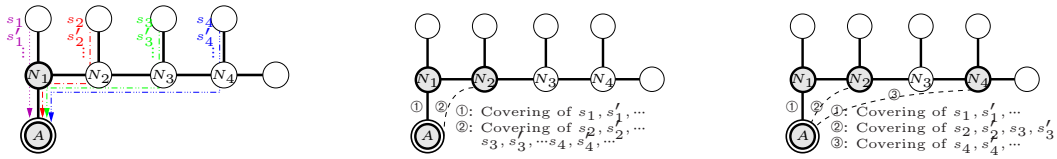
The algorithm to propagate subscriptions and place anchors works as follows: Subscriptions are issued by clients and are propagated throughout the network along the primary links *only*. Starting from the broker that a subscriber is directly connected to, each receiving broker stores a copy of the subscription in a set data structure called the *Master Subscription Table* (MST). Subscription entries in this set are in the form of $s = \langle id, preds, anchor \rangle$, where *id* is a unique identifier, *preds* are predicates specifying client interest, and *anchor* is the subscription anchor. As a subscription propagates in the network, we require its *anchor* to be adjusted at each intermediate broker as follows: If the issuing subscriber is within distance $\Delta - 1$, the *anchor* remains unchanged; otherwise, the *anchor* is set to the identifier of a broker which is one hop closer to the subscriber than the subscription's previous *anchor*. For example in Figure 3(a), the *anchor* of a subscription s issued by X_5 will be updated to N_4 before N_1 sends the message to Broker A ($\Delta = 4$). Note that the information to correctly adjust the anchors is already available locally at brokers as part of their MOM.

3.2 Working Subscription Table

In a similar manner that the WOM is constructed from MOM, brokers derive *Working Subscription Table* (WST) from their MST and use it for efficient publication forwarding. Construction of WST takes advantage of the information pre-computed in the WOM and takes place as follows: For each subscription, $s_{mst} \in MST$, a new subscription, s_{wst}, with an identical *preds* and *id* field but with an updated *anchor* is inserted into the WST. The *anchor* of s_{wst} is the identifier of the broker that $s_{mst}.anchor$ was substituted with (see edge contraction in Section 2.2). Figure **??** illustrates MST and WST of Broker A in a sample network. Subscription s_1's *from* field in Broker A's WST in Figure **??** points to Broker N_3. Subscription s_3's *from* field in Broker A's WST in Figure 3 points to Broker N_4. WST is reconstructed once the broker links change. This adapts routing tables to the current state of the overlay mesh. This process uses information that is available locally with no need to repropagate subscriptions or coordinate with neighbors.

3.3 Subscription Covering

Subscription covering [31, 7, 13] is an optimization technique intended to improve the performance of the matching operation. Subscription covering can be incorporated in our approach by having brokers compute separate



(a) Propagation of subscriptions takes place normally along primary links

(b) Covering of subs passing through N_2, N_3, and N_4 is computed together

(c) Each subscription can only be present in one covering set at a time

Figure 4: Covering is computed for subscriptions in WST with equal *from* fields.

covering relations for subscriptions that originated downstream each of their links. More precisely, brokers group subscriptions in the WST based on their *anchor*s. For each group, the covering predicate set that covers all subscription predicates is computed separately and used for matching. Evaluation of the covering predicates is normally more efficient and improves the matching performance. Figure 4 illustrates application of the subscription covering technique in a sample broker network. In this figure, Broker A stores subscriptions in its MST, but computes the covering relations based on the links it maintains (Figures 4(b) and 4(c)).

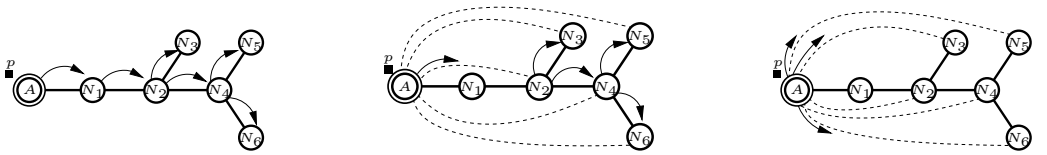
In Section 6, we experimentally study how broker configuration parameters impact the performance improvements gained via subscription covering.

4 Publication Forwarding

In this section, we elaborate on three publication forwarding strategies that differ in the way that brokers make use of available soft links. We use Figure 5 to describe how publication p is forwarded by Broker A in each strategy. In the figure, dashed lines represent A's available soft links and we assume that p matches subscriptions whose anchors point to N_3, N_5, and N_6.

- **Strategy 0** (S0): Publications are forwarded over primary links *only* and make no use of soft links. In Figure 5(a), A sends one copy of p to N_1. This strategy is identical to that of widely used conventional P/S systems [7, 12, 9] and is presented here only as the baseline for comparison.
- **Strategy 1** (S1): Publications are sent over both primary links and soft links such that primary paths to receivers do not overlap. In Figure 5(b), A sends p to N_2 (bypassing N_1). Broker N_2 will then be responsible to forward p to N_3 and N_4.
- **Strategy 2** (S2): Publications are sent on both primary links and soft links, but unlike S1, primary paths to receivers may also overlap. In Figure 5(c), A uses its soft links and directly forwards separate copies of p to N_3, N_5, and N_6.

Each strategy consumes different amount of output bandwidth per publication. For example, Broker A sends three copies of p in S2 while S0 and S1 each send only one copy. On the other hand, the total number of messages sent in S2 is less than S1 which is in turn less than S0. In what follows, we describe how to use WOM and WST to efficiently implement these strategies.



(a) S0: p is sent *6* times

(b) S1: p is sent *5* times

(c) S2: p is sent *3* times

Figure 5: Forwarding publication p matching subscribers at N_3, N_5, N_6 using different strategies. Primary and soft links are represented by solid and dashed lines, respectively.

Regardless of which forwarding strategy used, processing of publications involves two steps: *publication matching* and *path computation*. In the first step, brokers use their WST and a matching algorithm to identify the set of subscriptions that match the publication's content. The exact implementation of the matching algorithm is outside the scope of this paper and has been investigated extensively in the literature. We only assume that the output of the algorithm is in the form of a set of broker identifiers corresponding to the *anchor* of matching subscriptions in the WST. We use $Match(p)$ to denote this set for publication p. In the next step (i.e., path computation), the broker uses its WOM and computes a final set of neighbors to which it has a direct links. This set is denoted as $Fwrd(p)$, and once computed, the broker simply sends p over the corresponding links (if p matches a local subscription, the corresponding client receives a copy). In the remainder of this section, we present how $Fwrd(p)$ is computed.

4.1 Path computations for S0

This strategy concerns conventional P/S systems [7, 12, 9, 13] and is presented here as a baseline for comparison. Brokers do not establish and maintain soft links and Δ can effectively be set to 1. Furthermore, subscription anchors in WST only consist of immediate neighbors in the primary network. As a result, we have $Fwrd(p) = Match(p)$.

4.2 Path computations for S1

In this strategy, brokers possess and exploit soft links in order to bypass uninterested neighbors. At the same time, a forwarding broker intends to achieve this goal by sending the publication to the farthest reachable broker on the *intersection of the primary paths* to members of $Match(p)$. For efficient path computation, the broker takes advantage of the pre-computed auxiliary sets in its WOM as shown in Figure 6. In Lines 6–10, the set of brokers on the intersection of primary paths to mp is computed: $IntersectSet$. This is carried out by a series of set operations over the pre-computed auxiliary sets. The **while** loop of Lines 11–15 processes brokers, N_j, on the intersection of the paths in *descending order of distance*. If there is a broker in $Match(p)$ that is beyond N_j (i.e., $BeyondSet(A, N_j) \cap Match(p) \neq \varnothing$), then N_j is added to $Fwrd(p)$ and all brokers located beyond N_j (including N_j itself) are removed from $Match(p)$ (Line 14). This is correct since once N_j receives the publication, it sends it to all other downstream brokers. Finally, when $Match(p)$ becomes empty $Fwrd(p)$ is returned.

4.3 Path computations for S2

The goal of the broker in S2 (algorithms shown in Figure 7) is to directly send publications to all reachable anchors in $Match(p)$ excluding those that have a closer reachable broker on their primary path. This is different

```
1:  function COMPUTE_FORWARDING_S1
2:      Input Match(p)
3:      IntersectSet ←∅;
4:      Fwrd(p) ← ∅;
5:      j ← |BrokerArr_A|
6:      for all N_i ∈ Match(p) do
7:          if IntersectSet ∩ BetweenSet(A, N_i) = ∅ then
8:              IntersectSet ← IntersectSet ∪ BetweenSet(A, N_i)
9:          else
10:             IntersectSet ← IntersectSet ∩ (BetweenSet(A, N_i) ∪ BehindSet(A, N_i))
11:     while j > 0 ∧ Match(p) ≠ ∅ do
12:         if (IntersectSet ∩ BeyondSet(A, N_j)) ≠ ∅ then
13:             Fwrd(p) ← Fwrd(p) ∪ N_j
14:             Match(p) ← (Match(p) − BeyondSet(A, N_j))
15:         j ← j − 1
16:     return Fwrd(p)
```

Figure 6: Path computation for S1 at Broker A.

```
1: function COMPUTE_FORWARDING_S2
2:     Input Match(p)
3:     j ← 0; Fwrd(p) ← ∅
4:     while j < |Links| ∧ Match(p) ≠ ∅ do
5:         if N_j ∈ Match(p) then
6:             Fwrd(p) ← Fwrd(p) ∪ N_j
7:             Match(p) ← (Match(p) − BeyondSet(A, N_j))
8:         j ← j + 1
9:     return Fwrd(p)
```

Figure 7: Path computation for S2 at Broker A.

from S1 where publications are likely to be sent to pure forwarding brokers located on the intersection of paths to the anchors the path computation algorithm for S2. Intuitively, the brokers in $Match(p)$ are first considered in ascending order of distance (from lower bit-vector indices to higher). Each such broker is added to $Fwrd(p)$ (Line 6 in Figure 7), and all its downstream brokers are removed from $Match(p)$ (Line 7). Finally, when $Match(p)$ becomes empty $Fwrd(p)$ is returned.

4.4 Implementation note:

The size of all auxiliary sets is bounded by the number of brokers' links. Since this is relatively small, bit-vectors can provide an efficient implementation for the set operations in the algorithms: Broker N_j in WOM is associated with the j-th bit in a bit-vector and set union, and intersection operations are carried out via bit-wise '&' and '|'.

The following Lemmas and Theorem establish the correctness of strategies S1 and S2.

Lemma 1 *For every Broker A and neighbors B and B' in its overlay map, we have:*

$$B \notin BehindSet(A, B') \Rightarrow BetweenSet(A, B) \cap BetweenSet(A, B') \neq \emptyset$$

proof 1 *The proof follows from the construction of BetweenSet and BehindSet.*

Lemma 2 *If Broker A adds Broker B into IntersectSet during execution of Lines 6–10 of Figure 6 then:*

- *All Brokers in $BetweenSet(A, B)$ is also added to IntersectSet.*
- *If $B' \in BetweenSet(A, B)$ is the closest broker to A, then B' is never removed from IntersectSet.*

proof 2 *To prove the first part, observe that Broker B may only be added to IntersectSet in Line 8. At this point, all brokers in $BetweenSet(A, B)$ are also inserted into IntersectSet.*

To prove the second part, assume the contrary is true. Let $B' \in BetweenSet(A, B)$ be the closest node to A that is removed from IntersectSet during the k-th iteration of the **for** *loop (when Broker B_k is being considered). Removal of B' from IntersectSet can only happen in Line 10 provided that $B' \notin (BetweenSet(A, B_k) \cup BehindSet(A, B_k))$. However, this cannot happen due to Lemma 1.*

Lemma 3 *If Broker A executes Lines 6–10 of Figure 6 and B is an anchor in $Match(p)$, then either $B \in IntersectSet$ or there is another Broker $B' \in IntersectSet$ such that B' is on the primary path between A and B.*

proof 3 *Consider the contrary is true. The* **for** *loop of Lines 6–10 must have at some point considered B when iterating through $Match(p)$. At this point, if the* **if** *branch is executed then B is inserted into IntersectSet since $B \in BetweenSet(A, B)$. If the* **else** *branch is executed, then $\exists B' \in BetweenSet(A, B) \cap IntersectSet$. Once such B' is added into IntersectSet, Lemma 2 proves that it either stays in IntersectSet or there another broker $B'' \in BetweenSet(A, B')$ that is in IntersectSet at the end of the* **for** *loop. Either way contradicts the assumption.*

Theorem 1 *Forwarding strategies, S1 and S2 deliver publications to all matching subscribers.*

proof 4 *The proof is based on the fact that the P/S routing algorithm used in the primary network places subscription anchors in such a way that from any given Broker B, the sequence of anchors corresponding to matching subscriptions of a given publication message p constructs a directed connected path in the primary overlay network towards the issuing subscribers. The subscription propagation algorithm we presented in Section 3 obviously achieves this goal. As a result, strategy S0 delivers publications in the reverse paths of the propagation of matching subscriptions hop-by-hop towards the issuing subscribers.*

Strategies S1 and S2, on the other hand, both essentially do the same task while potentially skipping the uninterested brokers along the primary forwarding paths. Bypassing these pure forwarding brokers would not result in publication loss. Furthermore, at any point along these paths, the broker right after the skipped brokers that receives a copy of p can identify the sender as both the sending and receiving brokers are within Δ-neighborhoods of each other. What the receiving broker does with the publication message p is simply to re-do the process by identifying new subscription anchors and forwarding p even closer to the matching subscribers. Eventually p arrives at all brokers with local matching subscribers and will subsequently be delivered to the corresponding clients.

In the remaining of the proof, we needs to show that all matching subscription anchors in $Match(p)$ either directly receive p from a forwarding Broker A or A sends p to another Broker that is on the primary path between A and the anchors.

- *S1: By Lemma 3, after forwarding Broker A executes the* **for** *loop of Lines 6–10 in Figure 6, if B is an anchor broker in $Match(p)$, then either $B \in IntersectSet$ or there is another Broker $B' \in IntersectSet$ such that B' is on the primary path between A and B. In the first case, when the* **while** *loop of Lines 11–15 considers Broker B (*i.e. *$B_j = B \in Match(p)$), B is still in $Match(p)$ (otherwise, it implies that B has been removed from $Match(p)$ in Line 12 and following consideration of another broker B' by the* **while** *loop s.t. $B \in BeyondSet(A, B')$; this is impossible since it implies that B' is closer to A than B and violates the fact that the* **while** *loop iterates through brokers in descending order of their distance) and will be added to $Fwrd(p)$. In the second case, when the* **while** *loop considers broker B' (*i.e. *$B_j = B' \in Match(p)$), then it will receive p directly and will be responsible to further forward p (and send it to B if it has a matching subscription). Furthermore, Broker B will be removed from $Match(p)$ in Line 14. In both cases, B is removed from $Match(p)$ and the algorithm terminates.*
- *S2: The* **while** *loop of Lines 4- 7 in Figure 7 iterates through brokers in $Match(p)$ in increasing order of distance. Each broker $B \in Match(p)$ is removed from this set only if p is sent to B directly or if p is sent to another Broker B' s.t. $B \in BeyondSet(A, B)$.*

This concludes the proof.

5 Managing Broker Links

In large overlays, the overhead of establishing many connections makes it infeasible to maintain full network connectivity. We would thus like to allow brokers to selectively establish a small set of "good" soft links. A good link contributes most to the system performance and has two characteristics: First, it transmits a large volume of traffic, and second, it bypasses a large number of intermediate brokers in the primary network. In this section, we introduce *candidate* links and develop a profiling scheme to identify "good" links.

5.1 Soft Links and Candidate Links

A broker, say A, can have three types of links: *(i)* primary links ($prim_A$) are designated communication links in the primary network; *(ii)* soft links ($soft_A$) augment the primary network to a highly connected mesh overlay; and *(iii)* candidate links ($cand_A$) are not real communication links and only act as temporary stubs in the routing tables facilitating the process of identifying good soft links. We use the term broker *degree* to refer to the number of broker's primary links, and the term *fanout* for the maximum number of allowed communication links, i.e., primary and soft links combined. Finally, we use $Links_A$ to denote the set of all links at Broker A: $Links_A = prim_A \cup soft_A \cup cand_A$.

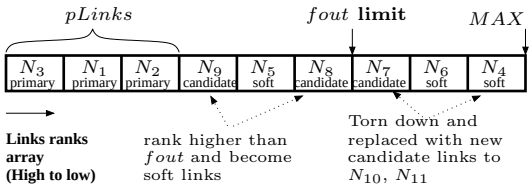


Figure 8: Ranked output of traffic profiling.

5.2 Publication Traffic Profiling

We define the *gain* of a link over time interval T to be the number of brokers that the link bypasses in the primary network times the number of publications that are sent over the link during T. More precisely, broker A computes the gain of its link to Broker N as follows:

$$gain = (\#\text{ pubs sent to } N \text{ during } T) * (dist(A, N) - 1)$$

Brokers use the gain function to rank their links in descending order (Figure 8) and links with higher ranks are preferred over those with lower ranks.

5.3 Candidate Links

Prospective soft links with unknown gains that are first considered for profiling are called candidate links. A candidate link acts as a stub in the broker's WOM and WST and enables publication profiling in a similar way to primary and soft links. In contrast to primary and soft links, however, a candidate link does not have a network connection and publications that are intended to be sent over a candidate link are transparently redirected over a primary or soft link to another neighbor that is closer in the primary network. This way, candidate links allow brokers to *locally* estimate their prospective gain without going through the link establishment process. Once a candidate link is determined to be "good", it is promoted to become a soft link and its connection is established.

5.4 Soft Link Management

Brokers periodically examine the gain of their links and determine which ones to keep and which ones to discard. The total number of links at each point is limited to a MAX number. This consists of at most *fanout* primary/soft links and (MAX – *fanout*) candidate links. All profiled links are ranked based on their measured gain (Figure 8). Soft links with low gains are discarded and candidate links with high gains are promoted to become soft links. In this process, brokers respect the *fanout* limit on their maximum number of communication links. Finally, brokers add new candidate links up to the MAX limit to be profiled in the next round. New candidate links are chosen based on the following heuristics: If an existing link to a neighbor, say X, has a high gain, then there is some chance that direct links to X's neighbors also achieve a good gain. This is especially true if X's high gain is due to the traffic that will eventually be forwarded to its neighbors. To determine such cases, the broker considers the neighbors of a high gain link as new candidate links. If such links indeed prove to deliver a good gain, they will be promoted to soft links in the future.

To adapt to network conditions, brokers exchange load information and measure links' round trip times. This information is used in the candidate selection process by prioritizing soft links that bypass slow links or overloaded neighbors. Such candidate links may not provide exceptionally good gains, nonetheless, brokers may choose to promote them to soft links if they have a minimum gain. This way, our approach enables brokers to effectively explore their neighborhoods in search of viable soft links. Once links are updated, the broker subsequently recomputes its WOM and WST.

5.5 Managing Primary Links

So far, we described how brokers choose their soft links based on gain, neighbors' load and network conditions. Addition and removal of soft links is a light-weight process and only requires local (not coordinated) routing table

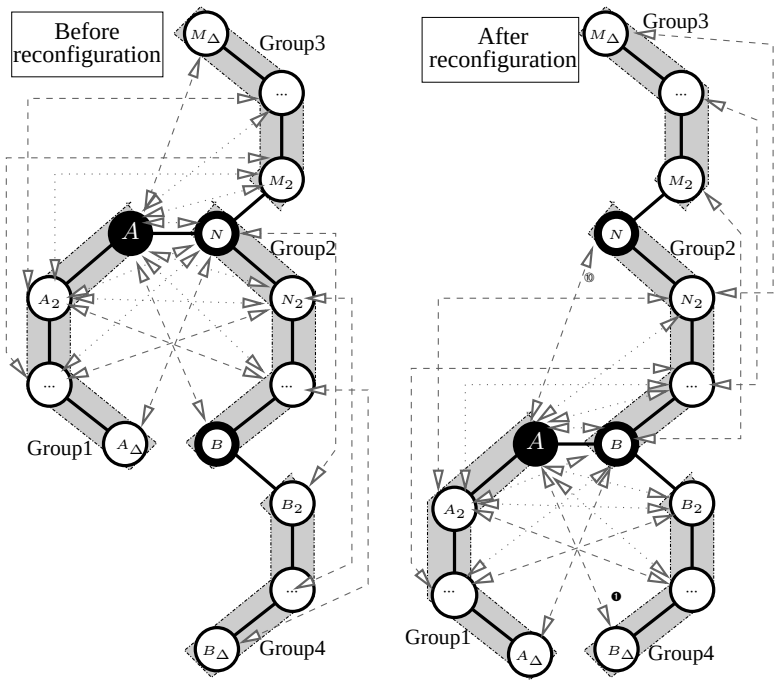


Figure 9: Overlay before (left) and after (right) Δ-move of Broker A. Solid lines are primary links and dashed lines represent anchor of subscriptions issued from affected parts of the network.

updates. Hence, brokers can afford to apply our scheme frequently and adapt swiftly to network and traffic changes. In contrast, primary links are meant to be more stable, mainly since changes to the primary network requires *costly coordinated updates* to MOMs and MSTs of many brokers.

Primary network reconfiguration is studied in the literature [29, 3, 19]. In short, these algorithms use different cost functions to determine which links to add or remove, and update the brokers' routing tables accordingly. To maintain generality and relevance of existing approaches to our work, in this section, we describe such primary network reconfiguration procedure in form of a Δ-move: A Δ-move is the process that a broker disconnects one of its primary links and establishes a new primary link to another broker within its original Δ-neighborhood. Any form of overlay modification can then be carried out via a series of Δ-moves, joins and departures (of edge brokers).

Definition 1 *In the Δ-move process of broker A, the* **moving path** $\mathcal{P}_{A:N\to B}$ *is the primary path between disconnecting Broker N where A disconnects from and Broker B where it connects to.*

Figure 9 illustrates the state of a network before (left) and after (right) Broker A Δ-moves from N to B. The solid lines in the figure represent primary links and the dashed arrows represent the anchor of subscriptions stored in the brokers' *MST*. Finally, $\mathcal{P}_{A:N\to B} = \langle N, N_2, \cdots, B\rangle$. We use this figure and describe how Broker A's and other nearby brokers' *Master Subscription Table*s are updated in a move process. To simplify description, we group A's neighboring brokers into four categories (based on their position relative to A's initial and final location):

- **Group 1:** Brokers in this group move with Broker A (e.g., $A_1 = A, A_2, \cdots, A_\Delta$ in Figure 9).
- **Group 2:** Brokers in this group lie on the primary path between the broker that Broker A disconnects from, i.e., N, and the broker that A connects to, i.e., B (e.g., $N_1, N_2, \cdots, N_i$ in Figure 9 where $i \leq \Delta$ and $N_i = B$). We refer to the ordered sequence of these brokers as the *moving path*. Based on the definition of a Δ-move, a moving path can be at most of length $\Delta - 1$. Thus, Brokers N and A are within each others' Δ-neighborhood.
- **Group 3:** This group includes those brokers that are within Δ-neighborhood of Broker A before the move but will *not* remain within Δ-neighborhood of Broker A after the completion of the move. In Figure 9, these brokers include $M_1, M_2, \cdots, M_\Delta$.

- **Group 4:** This group includes those broker that will *remain or enter* the Δ-neighborhood of A after completion of the move (excluding Group 2 brokers). In Figure 9, Group 4 brokers include $B_1, B_2, \cdots, B_\Delta$ where $B_1 = B$.

5.6 Updating Overlay Maps

Brokers in A's old and new Δ-neighborhoods receive A's new Δ-neighborhood as well as its moving path. With this knowledge, these brokers can compute changes to *their own* Δ-neighborhoods in a straightforward manner.

5.7 Updating Subscription anchors

Let $anchor_{(sid,X)}$ be the anchor of subscription with id sid stored at Broker X. During a Δ-move the Anchor field of subscriptions stored at brokers of Group 1 that point to Group 2,3,4 brokers need to be updated. Likewise, the anchor field of subscription entries stored at brokers in Group 2,3,4 that point to Group 1 brokers must be updated. There are 6 different cases to consider and we will discuss next in this section. Let $anchor'$ to represent the updated $anchor$ after the move takes place. Figure 9 illustrate the anchor of subscriptions at each broker before and after the move takes place (using arrows).

In the update process, Brokers A and B cooperate and compile a set of mappings that help brokers in Group 1,2,3,4 to update their anchors. The mappings are encoded as follows: $\{\mathbb{S}_{X,Y} \rightarrow Y\}$ where $X = \{A, B\}$ and $\mathbb{S}_{X,Y}$ is the set of subscription identifiers whose anchor stored at X point to Y. The set of subscription identifiers satisfy the following conditions:

- For Broker A, we have: $\mathbb{S}_{A,Y} = \{sid | Y = anchor_{(sid,A)}$ is in Group 1$\}$
- For Broker B, we have: $\mathbb{S}_{B,Y} = \{sid | Y = anchor_{(sid,B)}$ is in Group 2$\}$

These mappings along with A's original and final Δ-neighborhoods as well as its moving path allow brokers in all groups to recompute a new anchor for the subscriptions whose identifier appear in the above sets. For this purpose, each such broker must receive the following mappings:

- Broker X in Group 2,3,4 must receive $\{\mathbb{S}_{A,Y} \rightarrow Y\}$.
- Broker X in Group 1 must receive mappings from Broker B: $\{\mathbb{S}_{B,Y} \rightarrow Y\}$. Furthermore, $anchor_{(X,Z)}$ s that point to Broker Z in Group 2 are updated in a straightforward manner using the overlay neighborhood information and moving path (*i.e.*, there is no need for subscription identifiers).

5.8 Optimization

Transfer of the entire set of subscription identifiers might be too costly. Since these identifiers are already stored at Group 1,2,3,4 brokers, it is easier to use Bloom filters [**?**] to encode these sets, *i.e.*, $\mathbb{S}$. A receiving broker will then need to just iterate through its already stored subscription identifiers and test whether they are in the Bloom filter of each of the mappings. If so, then the broker proceeds to update the anchor of the subscription accordingly.

6 Evaluation

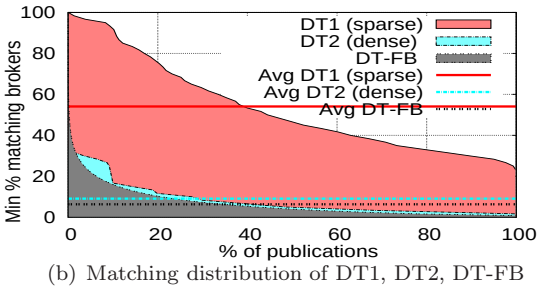
We carried out experimental evaluation on a cluster as well as the Planetlab testbed using Publiy, our Java-based open-source prototype implementation. We used several network configurations and workload datasets to compare S1 and S2 against S0 as baseline.

Experimental configurations: Our experimental setups varied in terms of *(i)* network size and layout; *(ii)* subscription matching distribution and density in workload datasets; *(iii)* parameters including Δ, *fanout*, and broker degree; and *(iv)* broker bandwidth capacity. These configurations are motivated by the use of large-scale P/S systems in datacenter or wide area environments and are summarized in Table 1. Throughout this section, we refer to these configurations using their short name, *e.g.*, C1, C2, etc.

Our cluster machines come in two variations: 8 core 2.66 GHz 64-bit Intel Xeon CPU with 8 GB of memory, or quad-core 1.86 GHz 64-bit Intel Xeon CPU with 4 GB of memory. Our Planetlab nodes ranged from single,

Dataset	Config.	Subs	Brokers with matching subs
DT1	C1, C4	6,500	8% (Sparse)
DT2		36,000	53% (Dense)
DT3	C2	30,000	9% (Sparse)
DT4	C5	900	9% (Sparse)
DT5		6,000	54% (Dense)
DT6	C3	10,000	4% (Covering)
DT-FB	C1	120,000	6.5% (Sparse)

(a) Datasets summary



(b) Matching distribution of DT1, DT2, DT-FB

Figure 10: Workload specification.

dual, and quad core Intel-family CPUs with 1.8 – 3.2 GHz and 1 – 3 GB of memory. Due to the shared nature of resources on Planetlab we configured brokers to use a maximum of 30 MB of memory.

Subscription Workloads: Figure 10(a) summarizes the matching distribution in our subscription and publication workload datasets. These workloads are either extracted from real-world traces of user interaction in social networking applications [30] (*i.e.*, dataset DT-FB) or synthesized based on Zipf distribution and come with sparse and dense matching densities. This choice was motivated by studies of Liu, *et al.* that showed that RSS feeds popularity follows Zipf distribution [18]. Figure 10(b) compares the matching density for datasets DT1, DT2 and DT-FB by illustrating what percentage of publications are delivered to what percentage of brokers with local matching subscribers. Finally, subscriptions in dataset DT6 have covering relationships.

Half of our brokers in each configuration host subscribers and the other half host publishers. In the beginning of each run, brokers propagate subscriptions in the network. In principle, each subscription can be issued by a separate subscriber process. However, due to scarcity of our resources running thousands of clients was infeasible. Instead, we skipped message delivery to clients and only considered publication forwarding within the broker overlay network. We thus had brokers log delivered publications to local files instead of sending them to an actual subscriber. We believe that this approach is fair for our comparative study, since delivery to local subscribers incurs exactly the same amount of processing and bandwidth in all strategies.

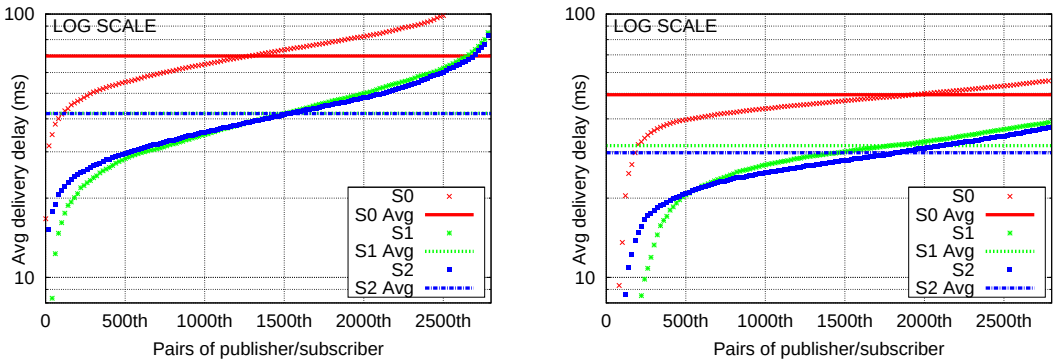
Publication delivery delay: Figure 11 illustrates the publication delivery delay using configuration C1 and *fanout* of 15. Aggregate publish rate is 1,800 msg/min and takes place over 5 minutes. At this rate, no network hotspots are formed and all strategies deliver the same number of messages. In Figure 11, each strategy's average publication delivery delay (vertical axis) is plotted in log scale for all pairs of publishing and subscribing brokers (horizontal axis). The average delivery delay for all receivers is also shown as horizontal lines in the graph. Figure 11(a) illustrates that for executions with sparse dataset DT1, the average delay lowers from 69.3 ms for S0 (baseline) to 42.0 ms for S1 and 41.8 ms for S2, a 40% improvement over the baseline.

Likewise, for executions with dense dataset DT2, the average delivery delay improves from 49.4 ms for S0 to 31.6 ms and 29.7 ms for S1 and S2, respectively (Figure 11(b)). This shows a 36% and 40% improvement for S1 and S2, respectively. These results indicate that S2 has a slight advantage over S1 in terms of publication delivery delay. At the same time, they both outperform S0 which incurs the highest delivery delay.

Publication forwarding path length: Publication hop count provides valuable insight into the internal workings of the system and has a direct impact on delivery delay, throughput and ultimately scalability. It is expected that if brokers maintain larger number of soft links, the created overlay mesh becomes more connected. This means

Config.	Net. size	Broker degree	Platform
C1	120	3	Cluster
C2	250	3	Cluster
C3	500	3	Cluster
C4	120	10	Cluster
C5	21	3	Planetlab

Table 1: Experimental configurations.



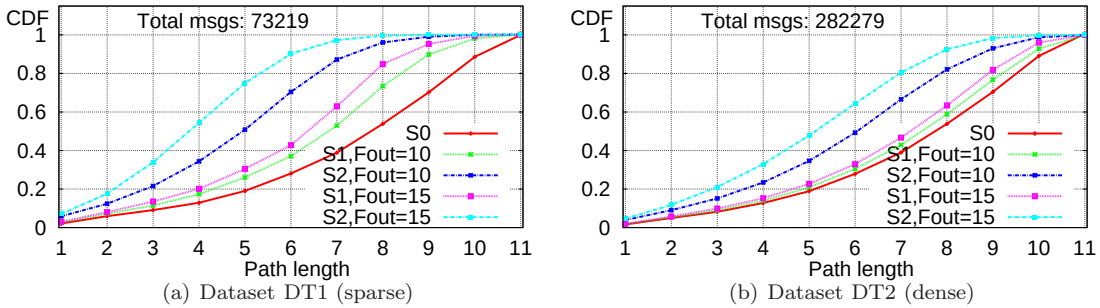
(a) C1, DT1, *fanout* = 15 (b) C1, DT2, *fanout* = 15

Figure 11: Publication delivery delay: x-axis shows pairs of source-destination brokers; y-axis is avg. delivery delay for corresponding pair. Horizontal lines show all-pair avg. delay.

that there are more alternative routes in the network, the best of which can be picked selectively to reduce the number of pure forwarding brokers. This subsequently lowers the publication hop count. We verified this trend experimentally. This is illustrated in Figure 12 which plots cumulative distribution function (CDF) of publication propagation path lengths. We used datasets DT1 (sparse) and DT2 (dense) and varied the broker *fanout* from 10 to 15 ($\Delta = 4$). The aggregate publish rate is low (1,800 msg/min) to ensure that all strategies deliver the same number of messages (about 73,000 publication deliveries for DT1 and 280,000 for DT2). The graphs show that S0 sends messages over the longest propagation paths while S1 and S2 shorten propagation paths substantially (data points move to the left). Furthermore, increasing *fanout* from 10 to 15 improves this trend and further shortens propagation path lengths.

Number of pure forwarding brokers: As we mentioned earlier, due to the selective nature of content-based matching, some brokers may inevitably relay publications that are not of interest to their local subscribers. Availability of a diverse set of alternative forwarding paths in our overlay mesh enables brokers to tailor the actual propagation path of publications based on the relative location of matching subscribers at runtime. Our measurements reported in Table 2 show that compared to S0, strategies S1 and S2 cut the number of pure forwarders by 53% and 68%, respectively. Furthermore, if we consider the *yield* of a P/S system as the total number of publications delivered (i.e., arrive at brokers with matching local subscribers) over the total number of publications sent between brokers (including those that are sent to pure forwarders), we see that S2 achieves a yield of up to 80%. This implies that only 20% of publications are sent to pure forwarders. In summary, using sparse workloads, strategies S1 and S2 achieve 19% and 27% improvement in yield over S0, respectively. For dense workloads, yield improvement over S0 is slightly lower: 14% for S1 and 21% for S2.

Network traffic: Fewer pure forwarders and higher yield means that the system can deliver the same number of publications by sending fewer messages among brokers. This lowers network traffic and improves efficient bandwidth



(a) Dataset DT1 (sparse) (b) Dataset DT2 (dense)

Figure 12: Publication propagation path lengths.

Delivered pubs	Strategy	Pure frwdrs	Yield
73,000 (Dataset: DT1)	S0	91,000	44%
	S1	42,000	63%
	S2	29,000	71%
284,000 (Dataset:DT2)	S0	195,000	59%
	S1	104,000	73%
	S2	69,000	80%

Table 2: Pure forwarding brokers and system yield.

use. Figure 13 illustrates the total network traffic in a run of C1 with an input publication rate of 1,800 msg/min (publication size is 1.2 KB). At this rate all strategies deliver the same number of messages. The average network traffic using S0, S1 and S2 is 1.98 MB/s, 1.66 MB/s and 1.52 MB/s, respectively. This indicates that S0 consumes 30% more traffic than S2 to deliver the same number of publications. The inner graph in Figure 13 shows the system's non-publication traffic consisting of load information messages exchanged between brokers. This overhead is exclusive to S1 and S2 only and amounts to merely 2% of total traffic.

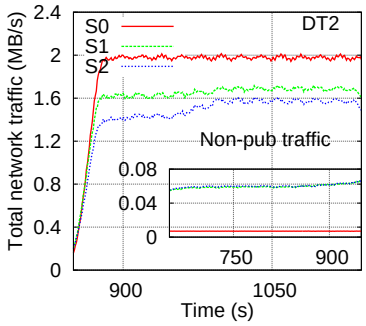


Figure 13: Network traffic.

CPU utilization: Strategies S1 and S2 update WOM and WST after changes to broker links (roughly every 20 secs in our implementation). Our measurements (shown in Figure 14) indicate that each update to WOM takes about 0.8 ms. Construction of WST, on the other hand, is dependent on the size of the subscription routing table and takes about 17 ms for a workload that consists of 6,500 subscriptions. Finally, thanks to our efficient use of bit-vectors for path computation, the time it takes for S1 and S2 to forward publications virtually remain unchanged.

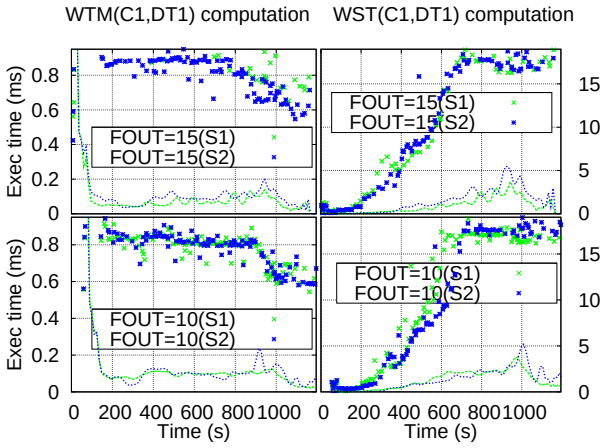
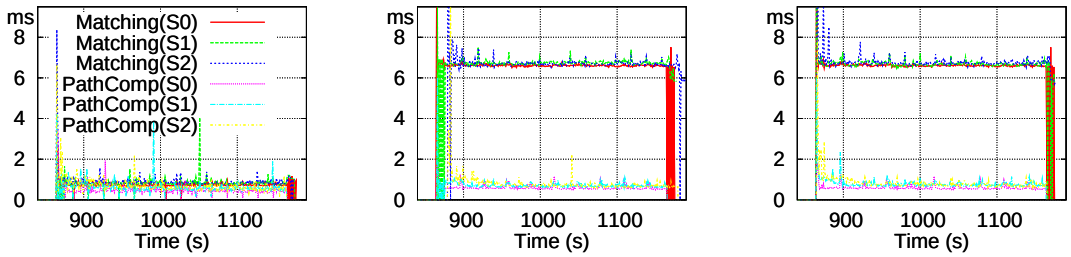


Figure 14: WOM and WST update times (C1, with (left) and DT2 (right)): x-axis is experiment time and y-axis is computation time. Dots represent individual updates and lines show averages.



(a) *fanout*10, DT1 (b) *fanout*10, DT2 (c) *fanout*15, DT2

Figure 15: Matching and path computation time (y-axis) over execution time (x-axis).

We now investigate the computational cost associated with our approach. Figure 15 illustrates the amount of time it takes to match each publication and compute the paths over which it must be forwarded. To eliminate any interference between processes, we ran a single broker on the cluster node that we performed these measurements on. In Figure 15(a), since the workload's subscription size is small (DT1) and the matching time is low (at about 1.8 ms/msg) and the path computation time is about 1 ms across all three strategies. In Figures 15(b) and 15(c), the matching time grows since workload DT2 has significantly more subscriptions. However, as seen in the figure, the path computation time remains *unchanged* for all strategies. Finally, as shown in Figures 15(b) and 15(c) increasing *fanout* from 10 to 15 has no impact on the path computation time. This is due to our efficient implementation of the working overlay map and working subscription tables that use bit vectors.

Impact of *fanout* on broker performance: Increasing *fanout* improves network connectivity in general but comes at the cost of maintaining a larger number of concurrent connections. Important contributing factors in this regard are buffer management and TCP's congestion control mechanism. Additionally, as we investigate in this section, a larger *fanout* limits the advantages brought about by subscription covering techniques and thus contributes to a degradation in matching performance. This effect is due to *fragmentation* of the subscription space as covering sets must be computed over a larger number of broker links. We clarify this point using a simple example: If subscription s_1 covers s_2 and s_2 covers s_3, then a broker that possesses only one link (*fanout* = 1) computes a covering set that contains only s_1. This means that publications must be matched against one subscription only. On the other hand, if the broker possesses two or more links, there is a high chance that the covering sets grow larger. For example, if s_1's anchor is downstream one link and s_2 and s_3's anchors are downstream of another link, then there are two covering sets each with one subscription. Hence, matching is roughly twice as time consuming. The exact size of covering sets, of course, depends on selectivity of subscriptions, their relative origin in the network. w.r.t. the broker's links. To further investigate this effect we carried out experiments using configuration C3 with 500 brokers and dataset DT6 with 10,000 subscriptions. The checkered bars in Figure 16 represent average size of covering sets over a 120s interval normalized over the case of *fanout* = 5. It is evident that increasing *fanout* increases the size of covering sets. Furthermore, larger covering sets translate to an even sharper increase in predicate matching operations per publication. For example, between *fanout* of 5 and 10 there is a 12% jump in covering set size and a 34% jump in the number of predicate matching operations. This difference is due to the fact that covering subscriptions are more generic and usually come with fewer predicates than covered subscriptions which in turn are more specific and contain more predicates.

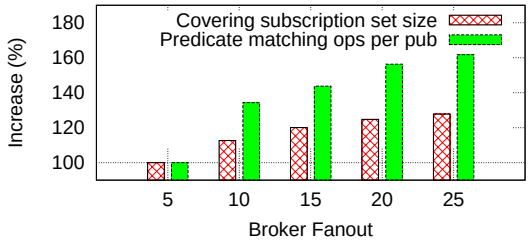
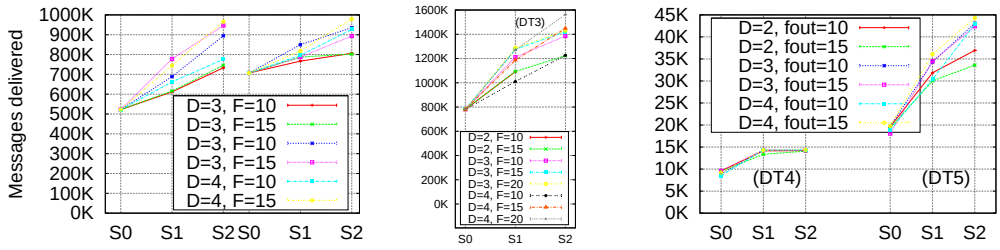


Figure 16: Higher *fanout* lowers gains of covering technique.



(a) Config. C1 (b) Config. C2 (c) Config. C5

Figure 17: System throughput (in the graph, 'D' and 'F' are Δ and *fanout*, respectively).

Overall system throughput: In large distributed messaging systems serving thousands of clients in a datacenter or an enterprise, throughput is perhaps the most important aspect of the system. In our approach, the reduced number of pure forwarders frees up brokers' valuable bandwidth and processing resources that can be used to disseminate larger number of publications. To study this, we carried out extensive throughput analysis with different configurations on a cluster and Planetlab. Figure 17(a) illustrates the throughput of configuration C1 under workloads DT1 (left graph) and DT2 (right graph). To measure the maximum attainable throughput, we used a high aggregate publish rate of 72,000 msg/min to push the system to the edge. At this rate, the number of publication deliveries within a fixed interval gives a clear understanding of the system throughput achievable in each strategy. Each data point in the figure is the *highest* delivered publication count in several executions with the designated parameters, i.e., Δ, *fanout*. Figure 17(a) illustrates that S2 delivers over 900,000 publications. This is about 90% more than the deliveries achieved with S0. Figure 17(b) attests that this improvement in throughput is also attainable in a larger overlay of 250 brokers (configuration C2). These results show a similar trend in which the number of deliveries grows by about 90% from around 800,000 to over 1.5 million messages.

Facebook dataset: We verified our results using datasets extracted from user interaction traces of online social networks [30]. We mapped users' friendship and interaction graphs of these traces into publication and subscription workloads, i.e., DT-FB To create this dataset, we assigned user identifiers to brokers and for each identifier we generated a subscription that represents the interest of the user in receiving activity of his friends. When a friend ,f, performs an action, e.g., posts a link or write to someone's wall, the broker to which f's identifier is assigned generates a publication containing information about the particular action performed as well as the identifier of f's friends in a (multi-valued) list predicate. This predicate matches the subscriptions issued by the corresponding users and the publication message is routed towards the brokers to which those friends' identifiers are assigned. According to our measures, this workload represents a sparse matching distribution with an average end-to-end message delivery to 6.5% of brokers. In a deployment with 120 brokers using C1 with aggregate publication rate of 3,600 msgs/min and system parameters of Δ of 4 and *fanout* of 20, our measurements indicate that S0, S1, and S2 achieve an overall throughput of 31,000, 43,000 and 67,000 messages in a 120 seconds interval. Compared to S0, this shows an improvement of 37% for S1 and 115% for S2.

Planetlab results: We performed experiments on the Planetlab in order to verify our results in a shared environment where CPU and bandwidth capacity is limited and variable. Figure 17(c) illustrates the results of our throughput analysis with configuration C5 and workloads DT4 (left graph) and DT5 (right graph). The measurement interval is 5 minutes, publication rate is 3,300 msg/min and the brokers have a capped bandwidth of 10 KB/s. At this rate, the system using DT4 and S0 becomes quickly congested and delivers fewer messages than expected. However, S1 and S2 both achieve full delivery goal (in the graph, data points for S1 and S2 fall on the same point of about 13,000 message deliveries). Similarly, for the case of DT5, S0 delivers the least number of publications. S1 and S2 on the other hand, achieve much higher (although not full) deliveries. The figures also illustrate that increasing *fanout* between 10 and 15 and Δ between 2, 3 and 4 progressively improves the achievable throughput.

The case of fanout = degree: As an alternative to our approach, one might suggest to use a high-degree broker overlay in which all communication links are primary links and Δ is 1. At the first glance, this might seem a compelling design since a high-degree overlay has low depth. But does this approach lead to a more scalable system? We considered this case by comparing configurations C1 and C4 which have broker degrees of

3 and 10, respectively. In both cases brokers possess the same number of communication links. Our throughput measurements indicate that C4 delivered 260,000 messages under workloads DT1. In comparison, the total number of deliveries in C1 and using S0 is 500,000 (note that as shown in Figure 17(a), S1 and S2 achieve much better results). Lower throughput of C4 is due to the fact that *internal brokers in a high-degree overlay pose a serious bottleneck* to a system's overall scalability and makes the network more prone to develop hotspots. In contrast, the use of soft links in S1 and S2 largely mitigates this problem since the soft link management process actively tries to bypass overly congested brokers. Moreover, the diversity of the set of available forwarding paths in our overlay mesh enables brokers to route messages around potential hotspots.

Forwarding strategy of choice: From a system-wide perspective, our results so far indicate that given any Δ and *fanout*, S2 outperforms S1 which in turn outperforms S0. But the question facing us now is whether there are any scenarios where the less performing strategies are preferred over S2. We now discuss such cases:

Case 1: Security or administrative concerns may prohibit unrestricted direct communication between arbitrary brokers. A simple scenario is when brokers belong to different administrative domains and in order to facilitate network management, authentication and enforce access control rules network operators require cross traffic to go through designated brokers. To comply with these restrictions, brokers resort to S0 and communicate over primary links only.

Case 2: In some settings, S1 may also be preferred over S2. For example, brokers may have different upload capacity and those with low bandwidth would like to preserve their available capacity as much as possible. In such cases, it would be beneficial for these brokers to selfishly resort to S1 which requires brokers to forward fewer messages per input publication (see Figure 5). In other words, such brokers can get by using less of their scarce bandwidth. We used C1 in two settings to investigate this effect experimentally. In one setting all brokers use S2 and in the other, one broker switches to S1 while other brokers continue to use S2. Using dataset DT2, we observed that the ratio of input to output bandwidth utilization at this broker reduces by almost 8% in the second setting. This ratio was below 0.8% when DT1 was used. Note that the reduction is more tangible under dense workloads since as we mentioned earlier S1 and S2 behave increasingly similarly under sparse workloads. Finally, the use of S1 by many brokers is likely to backfire and exhaust brokers' bandwidth earlier. This is due to the fact that widespread use of S1 can lead to a surge in all brokers' input traffic (see Figure 13).

7 Related Work

Our approach is inspired by Resilient Overlay Networks (RON) [2, 27] which use a link state routing technique atop *full-mesh* networks. Since RON aims to solve a generic unicast routing problem, its computed shortest paths is not optimal for multicasting publications to matching subscribers. In contrast, we aim to improve content-based routing of publications that are *sent to subsets* of interested subscribers. Furthermore, a full-mesh RON scales poorly while the use of Δ-neighborhoods in our approach, improves scalability.

Snoeren *et al.* [26] use a mesh network and forward publications over *multiple* disjoint paths. Their approach is concerned to exploit redundancy to guarantee delivery instead of achieving bandwidth efficiency. SplitStream [8] breaks down source data into multiple streams and propagates them over disjoint spanning trees. The goal is to reduce load on internal network nodes by spreading the traffic more evenly across the network. SplitStream is not readily applicable to content-based P/S due to the huge number of matching possibilities. Kyra [6] overcomes this problem by first slicing the subscription space into a manageable number of partitions and then building dissemination trees for each partition. Publications will be forwarded in the tree associated with the partition that they match. By considering each Kyra tree as a separate primary network, we can apply our approach and achieve adaptation and path redundancy within each tree. XNET [9] is another distributed P/S system which can directly take advantage of our approach.

Gryphon [4] uses the notion of Virtual Nodes (VN) in a tree-based overlay. Each VN consists of multiple replicas that construct alternative paths between upstream and downstream parts of the network. To mitigate an overloaded VN, Gryphon deploys an *extra* replica in that VN. As a result, an under-provisioned VN may easily become over-provisioned. In contrast, an overloaded broker in our approach simply informs its neighbors of its load condition and they create soft links to bypass it and lower its traffic. Overlay reconfiguration techniques [29, 3, 19] adapt the broker overlay by creating links between brokers with similar subscriptions. We can incorporate these techniques for primary network reconfiguration. However, changes to the primary network require costly coordinated updates to routing tables of many brokers. Our scheme to use soft links to adapt the overlay avoids the high cost of such

full reconfigurations.

In MEDYM [5], the firs broker computes a dissemination tree that only spans to brokers with local matching subscribers. The message piggybacks this tree and is used by other brokers in a source routing-like manner. This has the advantage that the propagation tree has *no pure forwarding brokers*. However, inclusion of source routing information incurs high overhead, especially for messages destined to a large number of brokers. Li *et al.* [16] create redundant forwarding paths in a cyclic overlay using overlapping advertisement trees. The quality and diversity of the paths, however, depend heavily on overlapping advertisements and their uncontrolled propagation patterns in the overlay. In contrast, our approach actively creates soft links to maintain redundant forwarding paths after taking broker load and links quality into account.

Semcast [22, 21] supports content-based data dissemination by first forwarding publications between brokers using channel-based techniques and then applying content-based filtering at edge brokers. Semcast's performance heavily depends on the workload and its channelization algorithm. If done poorly, filtering at edge brokers discards publications that have traversed long distances in the network. Apart from this, we believe application of our scheme to Semcast may not achieve as much improvement as in the general content-based case. This is due to the fact that publications must reach *all* channel brokers. As a result, there is no pure forwarding broker and our approach achieves no improvement by bypassing pure forwarders.

8 Conclusions

In this paper, we developed a novel approach to adapt the P/S overlay based on publication traffic and network conditions by selectively creating special links, called soft links. Soft links boosts the network connectivity and provide a large number of forwarding paths. Diversity of the end-to-end forwarding routes created in the overlay mesh is particularly suited for content-based P/S systems in which recipients of a given publication are not known in advance and only determined at runtime after subscription matching. Furthermore, thanks to the notion of Δ-neighborhoods our approach does not require coordinated route updates as each broker unilaterally decides which links to maintain. Our extensive experimental results carried out on a cluster and Planetlab confirm that our approach significantly improves system's throughput and efficiency.

References

[1] A. Adya, J. Dunagan, and A. Wolman. Centrifuge: integrated lease management and partitioning for cloud services. In *NSDI*, 2010.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. *Computer Communication Review*, 32(1), 2002.

[3] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. *Comput. J.'07*.

[4] S. Bhola, R. E. Strom, S. Bagchi, Y. Zhao, and J. S. Auerbach. Exactly-once delivery in a content-based publish-subscribe system. In *DSN*, 2002.

[5] F. Cao and J. Singh. MEDYM: Match-early and dynamic multicast for content-based publish-subscribe service networks. *ICDCSW*, 2005.

[6] F. Cao and J. P. Singh. Efficient event routing in content-based publish-subscribe service networks. In *IEEE INFOCOM 2004, Hong Kong*, 2004.

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

[8] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in cooperative environments. In *SOSP*. ACM, 2003.

[9] R. Chand and P. Felber. XNET: a reliable content-based publish/subscribe system. In *SRDS*, 2004.

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

[11] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *PVLDB*, 2008.

[12] G. Cugola and *et. al.* The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *Trans on Software Eng.*, 2001.

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

[14] Global Data Synchronization Network (GDSN). `http://www.gs1.org/docs/gdsn/gdsn_brochure.pdf`.

[15] Google Publish/Subscribe (GooPS), 2009. CANOE summer school.

[16] G. Li, V. Muthusamy, and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. In *Middleware*. Springer, 2008.

[17] G. Li, V. Muthusamy, and H.-A. Jacobsen. A distributed service-oriented architecture for business process execution. *TWEB*, 4(1), 2010.

[18] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *IMC*, 2005.

[19] M. Migliavacca and G. Cugola. Adapting publish-subscribe routing to traffic demands. In *DEBS*, 2007.

[20] B. Oki, M. Pjluegl, A. Siegel, and D. Skeen. The informationbus – an architecture for extensible distributed systems. In *SOSP*, pages 58–68, 1993.

[21] O. Papaemmanouil and U. Çetintemel. Semantic multicast for content-based stream dissemination. In *WebDB*, 2004.

[22] O. Papaemmanouil and U. Cetintemel. SemCast: Semantic Multicast for Content-Based Data Dissemination. In *ICDE*, 2005.

[23] Publish Subscribe Internet Routing Paradigm (PSIRP). `http://www.psirp.org/publications`.

[24] PubSubHubBub. `http://code.google.com/p/pubsubhubbub/`.

[25] M. Sadoghi, M. Labrecque, H. Singh, W. Shum, and H.-A. Jacobsen. Efficient event processing through reconfigurable hardware for algorithmic trading. In *VLDB'10*.

[26] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-based content routing using XML. In *SOSP '01*.

[27] D. Sontag, Y. Zhang, A. Phanishayee, D. G. Andersen, and D. Karger. Scaling all-pairs overlay routing. In *CoNEXT*, 2009.

[28] TIBCO Rendezvous. `http://www.tibco.com/products/soa/messaging/rendezvous`.

[29] A. Virgillito, R. Beraldi, and R. Baldoni. On event routing in content-based publish/subscribe through dynamic networks. In *FTDCS*, 2003.

[30] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *EuroSys*, 2009.

[31] Y. Zhao, S. Bhola, and D. Sturman. A general algorithmic model for subscription propagation and content-based routing with delivery guarantees, 2005.