Opportunistic Multipath Forwarding in Content-based Publish/Subscribe Overlays

Reza Sherafat Kazemzadeh[1] and Hans-Arno Jacobsen[2]

[1] reza@eecg.utoronto.ca
[2] jacobsen@eecg.utoronto.ca
Middleware Systems Research Group, University of Toronto

Abstract. Fine-grained filtering capabilities prevalent in content-based Publish/Subscribe (pub/sub) overlays lead to scenarios in which publications pass through brokers with no matching local subscribers. Processing of publications at these *pure forwarding* brokers amounts to inefficient use of resources and should ideally be avoided. This paper develops 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 network provides basic end-to-end forwarding routes, the mesh structure provides a rich set of alternative forwarding choices which can be used to bypass pure forwarding brokers. This provides unique opportunities for load balancing and congestion avoidance. Through extensive experimental evaluation on the SciNet cluster and PlanetLab, we compare the performance of our approach with that of conventional pub/sub 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 properties of the Publish/Subscribe (pub/sub) model has led to its adoption in a variety of enterprise, datacenter and wide-area network environments [1,2,3,4]. Microsoft, Google and Yahoo, for instance, use pub/sub for end-user notification delivery [5], data dissemination in large-scale server farms [3], and in distributed data storage systems [2]. In enterprise settings, pub/sub has appeared in several contexts and standards including Enterprise Service Bus (ESB) [6,4], algorithmic trading [7], WS-Notifications and WS-Eventing. In wide-area networks, pub/sub messaging has been used in push-based RSS feeds [8], global supply chain data exchange networks [1], and as a potential addressing and routing paradigm for future Internet protocols [9].

Widespread adoption of the pub/sub model further underlines the significance of scalable architectures that can efficiently utilize network resources in order to achieve low message delivery delay and high throughput. A distributed content-based pub/sub system deploys a set of dedicated application layer routers (*a.k.a.* brokers) to form an overlay network [10,11,12,13,14,15,16]. Clients connect to brokers and are offered the flexibility to specify fine-grained filtering constraints on publications they are interested to receive. Publications

that satisfy these constraints are forwarded through the overlay towards brokers where interested subscribers reside and are then delivered to those clients.

It is generally infeasible to maintain full connectivity in a large overlay and it becomes imperative to only utilize a selective set of all links. Furthermore, in content-based pub/sub systems the set of matching 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 pub/sub systems use a shared dissemination overlay and may forward publications through uninterested (*a.k.a. pure forwarding*) brokers with no local matching subscribers. Processing of publications at pure forwarding brokers increases publication hop count and propagation latency, and therefore amounts to inefficient use of bandwidth and computational resources in the network.

To lower the number of pure forwarders, reconfiguration techniques modify the overlay step-by-step by adding links between brokers with similar subscriber interests and removing links between those with less similarity [17,18]. Altering overlay links in this manner has a large system-wide footprint and while beneficial to some end-to-end publication flows it can at the same time be detrimental to many others. Furthermore, each reconfiguration step requires coordinated updates to routing tables of many brokers, a process that is slow, costly and potentially disruptive. Alternatively, clustering techniques group subscribers with similar interests and move them closer to brokers where publishers reside [19]. In content-based pub/sub systems in which clients may have widely varying interests, the performance of clustering schemes is not always guaranteed. Furthermore, clustering algorithms may prescribe clients with multiple subscriptions to connect to more than one broker, an inconvenience that must ideally be avoided.

A related common problem in virtually all existing pub/sub systems is that overlay forwarding paths are set up in a *fixed end-to-end manner*. In other words, *a publication is forwarded over a fixed path to a destination broker (where matching subscribers are connected) regardless of whether or not the message is of interest to subscribers at intermediate brokers along the path*. This rigidity inevitably results in a large number of pure forwarders, especially in a content-based pub/sub system with highly varied subscriber interests. This deficiency could be mitigated if the overlay connectivity between sources and destinations offered a multitude of redundant forwarding paths giving brokers the flexibility to pick the best forwarding path for each publication on a *one-by-one* basis.

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 end-to-end 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 collectively construct a highly connected overlay mesh and provide a rich set of redundant paths between all source and destination broker pairs. 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 distinct forwarding paths among them. This is increased to more than 1,000 for 13% of brokers in a network of size 250. In our approach, all these routes are readily available for publication forwarding and the decision on which path to take is made at runtime and based on the relative location of matching subscribers. This is in contrast to most pub/sub systems [13,10,14] which use *fixed end-to-end forwarding paths* and send a message towards 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.

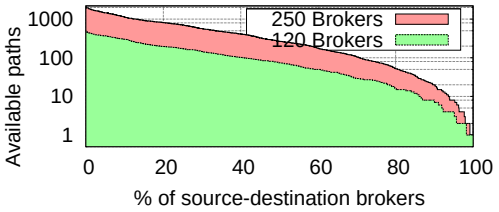


Fig. 1: Number of end-to-end forwarding paths in networks of 120 and 250 brokers (log scale).

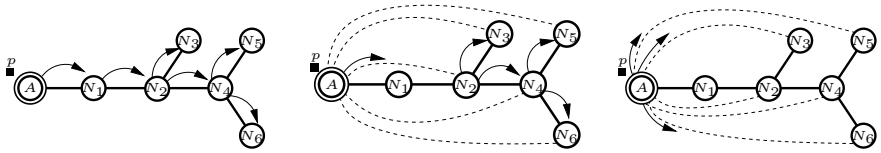
To forward publications in the overlay mesh and determine the relative location of matching subscribers, brokers rely on knowledge of their neighboring brokers within a certain distance, denoted by configuration parameter Δ. This knowledge enables a broker to anticipate how publications flow within its 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. As another advantage of our approach, modifying the overlay mesh of soft links requires only local updates to routing tables. This is a light-weight process and a significant improvement over full overlay reconfiguration techniques [17,18] 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 overlay mesh for pub/sub systems; *(ii)* techniques to update brokers' routing tables based on network connectivity with no need for coordination among neighbors; *(iii)* four 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 evaluations using a Java-based open-source implementation, called *Publiy* [20].

2 Publication Forwarding Strategies

In this section, we give a high level overview of four publication forwarding strategies that we develop in this paper. We defer the details of how each forwarding strategy can be efficiently implemented to subsequent sections.

We use Figure 2 to describe how publication p is forwarded by Broker A in each strategy. In the figure, p matches subscribers local to Brokers N_3, N_5, and N_6. Moreover, solid lines represent primary links in the network and dashed lines are extra soft links created and maintained in our approach.



(a) S0: p is sent *6* times (b) S1: p is sent *5* times (c) S2: p is sent *3* times

Fig. 2: 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.

Strategy 0 (S0): This strategy corresponds to conventional pub/sub systems and is presented here mainly as baseline for comparison. Conventional approaches are based on fixed end-to-end forwarding paths [13,14,16] along which matching publications traverse towards interested subscribers. These paths are established over communication links which we refer to as primary links. In Figure 2(a), Broker A sends one copy of p to N_1 which is its immediate neighbor located closer to matching subscribers. Observe that since forwarding paths are maintained in a fixed end-to-end manner, Broker A's decision to send p to this neighbor is not impacted by whether p is of interest to intermediate brokers along p's projected downstream propagation path.

Strategy 1 (S1): Brokers using this strategy take advantage of their neighborhood knowledge to *anticipate the propagation path of publications within their Δ-neighborhoods*. Armed with this knowledge, brokers identify nearby pure forwarders and attempt to bypass them using additional soft links that they possess. S1 allows brokers to utilize both primary links as well as soft links and send publications to their farthest reachable neighbors that are on the intersection of primary paths towards all matching subscribers. In Figure 2(b), the intersection of primary paths from Broker A to Brokers N_3, N_5 and N_6 consists of path $\langle N_1, N_2 \rangle$. As a result, Broker A sends p to N_2 (*i.e.*, the farthest reachable neighbor on this path) thus bypassing N_1. Broker N_2 will then be responsible to forward p to N_3 and N_4. Observe that S1 enables brokers to improve the system's performance by opportunistically bypassing some pure forwarders.

Strategy 2 (S2): This strategy also makes use of both primary and soft links to bypass pure forwarding brokers but compared to S1 this is done in a more aggressive manner. More specifically, brokers that run S2 attempt to directly forward publications towards matching subscribers using their farthest reaching primary and soft links. In Figure 2(c), Broker A uses its soft links and directly forwards separate copies of p to N_3, N_5, and N_6. Note that this strategy improves the chance of bypassing a larger number of pure forwarders. However, this comes at the cost of having Broker A send more copies of publication p.

Strategy Hybrid (SH): The above strategies consume different amount of output bandwidth per processed publication, *e.g.*, Broker A sends three copies of p using S2 while S0 and S1 each require A to send only one copy. Brokers using SH take advantage of this trade-off and dynamically switch between S1 and S2 to tune their output bandwidth consumption. More specifically, a broker with limited output bandwidth uses S1 to minimize utilization of its scarce resources. This, however, can potentially lead to increased network-wide traffic.

On the other hand, brokers with no resource constraints use S2 which incurs fewer overall network messages at the expense of more bandwidth utilization at each forwarding broker per publication.

We underscore that the advantages of our forwarding strategies grow in content-based pub/sub systems featuring *selective multicast*. In these systems, publications are likely to match highly varied subsets of subscriptions (the number of these subsets grows exponentially with the system size). Furthermore, interested subscribers are not known in advance and are identified only at runtime. This high degree of unpredictability and matching diversity makes it inherently difficult to optimize the pub/sub overlay when forwarding paths are constructed in a fixed end-to-end manner (*i.e.*, the case of conventional pub/sub systems). In contrast, the flexibility of forwarding publications through a well-connected overlay mesh of soft links in our approach greatly remedies this problem and enables fine-grained tuning of forwarding paths in an *opportunistic manner*.

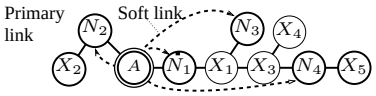
Realization of our forwarding strategies requires pub/sub brokers to maintain Δ-neighborhood knowledge and perform complex path computations in order to decide how to forward a publication. In the following sections, we elaborate on efficient techniques and data structures to enable brokers to harvest the benefits of high connectivity in the overlay mesh at a negligible overhead.

3 Overlay Maps

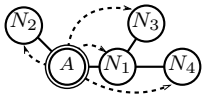
We assume that there is an initial pub/sub overlay that provides basic connectivity among brokers. We refer to this overlay and its links as the *primary network* and *primary links*, respectively. In this section, we elaborate on brokers' internal data structures, namely the *Master Overlay Map* and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

Master overlay maps: 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 includes neighboring brokers and their primary links located within distance Δ (i.e., Δ-*neighborhood*). The primary network is stable and changes infrequently, possibly due to occasional broker joins or departures. These changes are propagated hop-by-hop in Δ-neighborhoods so that nearby brokers update their MOMs accordingly.

Working overlay maps: 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 publication traffic, network and load conditions (details described in Section 6). 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 MOM and transforms the broker's initial knowledge of the primary network into a concise representation in accordance to its current



(a) MOM: A's primary view.



(b) WOM: A's mesh view.

Fig. 3: Master and working overlay maps at Broker A ($\Delta = 4$).

Neighbors	Substituted neighbors	Between set	Behind set	Beyond set
N_1	$\{X_1, X_3, X_4\}$	$\{N_1\}$	$\{N_2\}$	$\{N_1, N_3, N_4\}$
N_2	$\{X_2\}$	$\{N_2\}$	$\{N_1, N_3, N_4\}$	$\{N_2\}$
N_3	$\emptyset$	$\{N_1, N_3\}$	$\{N_2\}$	$\{N_3\}$
N_4	$\emptyset$	$\{N_1, N_4\}$	$\{N_2\}$	$\{N_4\}$

Fig. 4: Information in A's WOM ($\Delta = 4$).

set of primary, soft and candidate links. WOM is *reconstructed locally upon every update 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.

Step 1: Initially, A's WOM contains all the neighboring brokers and primary links in its MOM, in addition to A's own non-primary links. Since Broker A is often not directly connected to a number of its neighbors, the goal of this step (called edge contraction) is to weed out these neighbors from A's view – thus making it more concise. For this purpose, Broker A considers its neighboring brokers in descending order of distance in the primary network. For each Broker v with no direct link, the edge between v and a closer Broker w is removed and v is substituted with w. Furthermore, all edges incident (attached) to v are removed and attached to w. Once complete, *WOM forms a graph that only contains neighboring brokers to whom A maintains a direct link.* Figures 3(a) and 3(b) illustrate A's MOM and WOM before and after this process, respectively.

Step 2: Broker identifiers in the resulting graph are sorted in an array, denoted by $BrokerArr_A$, in ascending order of their distance in WOM. Henceforth, N_i, refers to the i^{th} broker in this array and we have $dist(N_i) \leq dist(N_{i+1})$.

Step 3: For each Broker N_i, three *auxiliary sets* are computed that contain identifiers of neighbors located in different relative positions with respect to A and N_i (see Figure 4). 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, $BetweenSet(A, N_4) = \{N_1, N_4\}$.
- $BeyondSet(A, N_i)$ 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, $BehindSet(A, N_1) = \{N_2\}$.

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

4 Subscription Routing Tables

Brokers maintain their subscription routing tables in a similar manner as to 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 construct end-to-end forwarding paths in the primary network. The entries in the latter, however, adapt these paths to the current overlay mesh connectivity as reflected in the broker's WOM.

Master subscription tables: 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 (the anchor of a local subscriber 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 in this manner is that brokers are able to anticipate the propagation path of matching publications within their Δ-neighborhoods and foresee forwarding paths towards all matching subscribers inside and outside of their neighborhoods. Availability of this information allows brokers to choose the actual publication forwarding path from a wealth of alternative routes within their neighborhoods (strategies that are used for this purpose are discussed in Section 5).

The anchor placement algorithm works as follows:[3] Subscriptions are issued by clients and are propagated throughout the network along the primary links *only*. Starting from the broker that a subscriber is 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 $s = \langle id, preds, anchor, ppath \rangle$, where *id* is a unique subscription identifier, *preds* are predicates specifying client interests, *anchor* is the subscription anchor, and *ppath* is the propagation path of the subscription *through the primary network*. As a subscription propagates in the primary 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 the 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 subscription to Broker A ($\Delta = 4$). Note that the information to correctly adjust the anchors is readily available locally at brokers as part of their MOM.

Working subscription tables: Similar to how WOM is derived from MOM, brokers derive the *Working Subscription Table* (WST) from their MST and use it for publication forwarding. Construction of WST uses information pre-computed in WOM as follows: For each subscription, $s_{mst} \in MST$, a new subscription, s_{wst}, with identical *preds* and *id* fields but with an updated *anchor* is added into the WST. The *anchor* of s_{wst} is the identifier of the broker that $s_{mst}.anchor$ was substituted with during Step 1 in the construction of WOM (see Section 3). WST is updated upon every change to brokers' links and in order to adapt routing tables to the state and connectivity of the overlay mesh. However, once constructed, all subsequently issued subscriptions (and unsubscriptions) are simultaneously added to (and removed from) both MST and WST.

[3] Applicability of our approach extends to other pub/sub routing schemes that can accommodate the placement of subscription anchors in brokers' Δ-neighborhoods.

Subscription covering: Subscription covering is an important optimization technique that improves matching performance by compacting brokers' routing tables. Subscription s_1 is said to *cover* s_2, *iff*, all publications that match s_2 also match s_1. We compute covering sets for subscriptions with identical anchors. A publication is forwarded over a link if it matches at least one of the subscriptions in the corresponding covering set. Section 7 investigates the impact of broker parameters on performance gains brought about by the covering techniques.

5 Publication Forwarding

This section presents efficient techniques to realize the publication forwarding strategies of Section 2. Regardless of the exact strategy used, the 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 $Anchors(p)$ 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 direct links. This set is denoted as $Fwrd(p)$, and once computed, the broker forwards p accordingly (if the publication matches a local subscription, the issuing subscribers receive a copy of the publication). This section presents how $Fwrd(p)$ is computed.

Path computations for S0: This strategy concerns conventional pub/sub systems [13,14,16,10] and is presented here purely 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 $Fwrd(p) = Anchors(p)$.

Path computations for S1: In this strategy, brokers possess soft links and exploit them in order to bypass uninterested neighbors. At the same time, a forwarding broker attempts to achieve this goal by sending the publication to the farthest reachable brokers on the *intersection of the primary paths* to neighbors in $Anchors(p)$. For efficient path computation, the broker takes advantage of the pre-computed *auxiliary sets* in its WOM as shown in Figure 5. In Lines 4–8, the set of brokers on the intersection of primary paths to $Anchors(p)$ is computed: $Intersection$. This is carried out via a series of set operations over the pre-computed auxiliary sets. Next, the **while** loop in Lines 10–14 processes brokers, N_j, on the intersection of the paths in *descending order of distance*. If there is a broker in $Anchors(p)$ that is beyond N_j (*i.e.*, $BeyondSet(A, N_j) \cap Anchors(p) \neq \emptyset$), then N_j is added to $Fwrd(p)$ and all brokers located beyond N_j (including N_j itself) are removed from $Anchors(p)$ (Line 13). The rationale behind this is once N_j receives the publication, it sends the publication to all other downstream brokers, *i.e.*, $BeyondSet(A, N_j)$. Finally, when $Anchors(p)$ becomes empty all matching subscription anchors have been accounted for and $Fwrd(p)$ is returned.

Path computations for S2: The goal of brokers in S2 is to directly send publications to all reachable anchors in $Anchors(p)$ excluding those that have a

```
1: function COMPUTE_FORWARDING_S1(Anchors(p)) ▷Input: p's matching anchors.
2:     Anchors(p) ← {X ∈ Anchors(p)∧ primary path to X does not intersect with
   primary propagation path of p}                    ▷ Only keep downstream anchors.
3:     Intersection ←∅; Fwrd(p) ← ∅
4:     for all (N_i ∈ Anchors(p)) do
5:         if (Intersection ∩ BetweenSet(A, N_i) = ∅) then
6:             Intersection ← Intersection ∪ BetweenSet(A, N_i)
7:         else
8:             Intersection ← Intersection ∩ (BetweenSet(A, N_i) ∪ BehindSet(A, N_i))
9:     j ← |BrokerArr_A|
10:    while (j > 0 ∧ Anchors(p) ≠ ∅) do
11:        if (Anchors(p) ∩ BeyondSet(A, N_j) ≠ ∅ ∧ N_j ∈ Intersection) then
12:            Fwrd(p) ← Fwrd(p) ∪ {N_j}
13:            Anchors(p) ← Anchors(p) − BeyondSet(A, N_j)
14:        j ← j − 1
15:    return Fwrd(p)
```

Fig. 5: Path computation for S1 at Broker A.

closer reachable broker on their primary path. This is different from S1 where publications are likely to be sent to pure forwarding brokers located on the intersection of paths to the anchors. Figure 6 presents the path computation algorithm. Intuitively, the brokers in $Anchors(p)$ are considered in ascending distance (*i.e.*, from lower subscripts to higher). Each such broker is added to $Fwrd(p)$ in Line 6, and all its downstream brokers are removed from $Anchors(p)$ in Line 7. Finally, when $Anchors(p)$ becomes empty $Fwrd(p)$ is returned.

Path computations for SH: Brokers that use the hybrid strategy SH monitor their output publication traffic and use a threshold (*e.g.*, 80% of their uplink capacity) to decide when to switch between S1 and S2. Once the link utilization passes this limit, the broker uses S1 to preserve its bandwidth. If the link utilization is lower than the limit brokers use S2 which more aggressively attempts to bypass pure forwarding brokers.

Implementation notes: 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_i in WOM is associated with the i-th bit in a bit-vector and set union, and intersection operations are carried out via bit-wise '&' and '|', respectively.

```
1: function COMPUTE_FORWARDING_S2(Anchors(p)) ▷ Input: p's matching anchors.
2:     Anchors(p) ← {X ∈ Anchors(p)∧ primary path to X does not intersect with
   primary propagation path of p}                    ▷ Only keep downstream anchors.
3:     j ←1 ; Fwrd(p) ← ∅
4:     while (j ≤ |Links| ∧ Anchors(p) ≠ ∅) do
5:         if (N_j ∈ Anchors(p)) then
6:             Fwrd(p) ← Fwrd(p) ∪ {N_j}
7:             Anchors(p) ← Anchors(p) − BeyondSet(A, N_j)
8:         j ← j + 1
9:     return Fwrd(p)
```

Fig. 6: Path computation for S2 at Broker A.

6 Managing Broker Links

In large overlays, the overhead of establishing many connections makes it infeasible to maintain full network connectivity. We would therefore like to have brokers selectively establish a small set of "good" soft links. A good soft link contributes most to the system performance and has three characteristics: First, it transmits a large volume of traffic; second, it bypasses a large number of intermediate pure forwarding brokers in the primary network; and, third, the more overloaded the bypassed brokers are the better a soft link is. In what follows, we first introduce *candidate* links and then devise a profiling scheme to identify "good" soft links.

A broker, say A, can have three types of links: *(i)* primary links (denoted by $pLinks_A$) are designated communication links in the primary network over which end-to-end forwarding paths are constructed; *(ii)* soft links (*i.e.*, $sLinks_A$) augment the primary network and build a highly connected mesh overlay; and *(iii)* candidate links (*i.e.*, $cLinks_A$) are not real communication links and only act as temporary stubs in the routing tables to facilitate the process of identifying good soft links. We use the term broker *degree* to refer to the number of primary links a broker has, and the term *fanout* for the maximum number of communication links, i.e., primary and soft links combined. Finally, we use $Links_A$ to denote the set of all links at Broker A, *i.e.*, $Links_A = pLinks_A \cup sLinks_A \cup cLinks_A$.

Publication traffic profiling: We define the *gain* of a link over time interval T as 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 times a scaling parameter that factors in the load of bypassed intermediate neighbors. More precisely, Broker A computes the gain of its own link to Broker N as follows:

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

The $loadScalingFactor$ is intended to further boost the gain of a link that bypasses overloaded intermediate brokers. It is defined as follows:

$$loadScalingFactor(N) = \Pi_{\forall X}(1 + min(0, normalizedLoad(X) - loadThreshold))$$

where $normalizedLoad(X)$ is the normalized load of Broker X located on the primary path between A and N. We considered output publication rate as our preferred broker load metric as opposed to input publication rate. This is due to the fact that a portion of brokers' input publication traffic is destined to local subscribers and cannot be avoided. In contrast, the output publication traffic is more indicative of the volume of publications that a broker relays. Relayed traffic can be opportunistically reduced using soft links that bypass pure forwarders.

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 funnelled over a primary or soft link to another neighbor that is closer in the primary network. Candidate

links allow brokers to locally estimate their 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 corresponding connection is established.

Soft link management: Brokers periodically examine the gain of their links and decide which ones to keep and which ones to discard. The total number of links in each round is constrained by the configuration parameter *maxlinks* and consists of at most *fanout* primary/soft links and ($maxlinks - fanout$) candidate links. Broker links are ranked based on their measured gains. 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 may add new candidate links 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 relayed 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 rounds.

To adapt to network conditions, brokers exchange load information and measure their communication links' round trip times. This information is used in the candidate selection process by prioritizing soft links that bypass slow links and overloaded neighbors. This simple cost model effectively enables brokers to explore their neighborhoods in search of viable soft links at a low cost.

Primary link management: The techniques presented so far enable brokers to choose their soft links based on publication traffic and neighbors' load conditions. Addition and removal of soft links is a light-weight process and only requires local (not coordinated) routing table updates. Hence, brokers can afford to employ this technique frequently and adapt swiftly to network and traffic changes. The soft link management scheme may also allow brokers to deal with transient crash or temporary disconnection of their neighbors [21]. In contrast, primary links are meant to be more stable, mainly since changes to the primary network require *costly coordinated updates* to MOMs and MSTs of many *affected* brokers (this cost is unavoidable in the case of permanent crash or departure).

We now present the primary network reconfiguration procedure in the form of a Δ-*move* whereby 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. Figure 7 illustrates the state of the primary network before (left) and after (right) Broker A moves from N to B. Solid lines in the figure represent primary links and the *move path*, $\mathcal{P}_{A:N\to B} = \langle N, N_2, \cdots, B\rangle$, is the primary path between Broker N and B. The figure also illustrates subscription anchors in nearby brokers' MSTs (*i.e.*, dashed arrows) that are affected by the move. We use Figure 7 to describe the move procedure via which Broker A's and other nearby brokers' *MOM*s and *MST*s are updated. Note that the update process only concerns the *master data structures*. Furthermore, in order to ensure

that the state of the network remains consistent, the move process uses an external coordinator to prevent concurrent moves from taking place within overlapping neighborhoods. Also, note that while a move is pending, the primary network can still enjoy a high level of adaptation that is brought about using soft links.

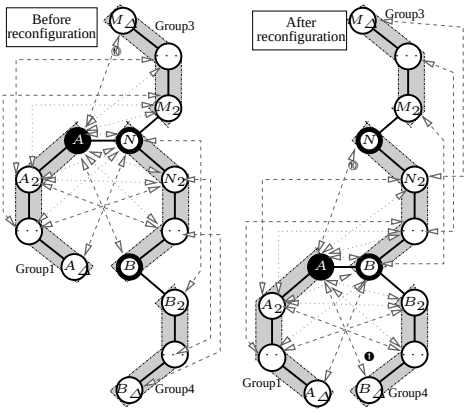


Fig. 7: Overlay before (left) and after (right) Δ-move of Broker A. Solid and dashed lines are primary links and subscription anchors at affected brokers, respectively.

Updating MOMs and MSTs: Following a move by Broker A from N to B, brokers within the old and new Δ-neighborhood of A must update their MOMs as well as the subscription anchors in their MSTs accordingly in order to correctly reflect the state of the primary network following the move. These brokers are said to be *affected* by A's move and can carry out updates as follows: An affected broker needs three pieces of information to compute its new MOM: Its initial MOM, as well as Broker A's initial and final MOMs. The update is done by excluding neighbors of A that are no longer within distance Δ and adding ones in A's new Δ-neighborhood that fall within distance Δ. Likewise, an affected broker needs three pieces of information to compute new anchors for the subscriptions in its MST. The information needed includes: Its old and new MOMs (the new MOM is computed as discussed above) as well as the move path $\mathcal{P}_{A:N\to B}$. The move procedure, described next, ensures that the information required in the above update process (*i.e.*, the move path and Broker A's old and new MOMs) is provided to affected brokers. As a result, affected brokers can locally compute their new MOMs and MSTs that reflect the state of the network after the join.

The Δ-move procedure: The move procedure involves the following phases:

Preparation phase: Before starting a move, Broker A contacts the coordinator for permission. Once granted, it contacts the destination Broker B and receives B's MOM. This will be used by A to construct temporary MOM_{tmp} and MST_{tmp} that reflect the state of its post-move Δ-neighborhood and subscription anchors.

Initialization phase: Broker A injects a special *move initiation message*, m_{init}, at Broker N. The move initiation message is propagated within Broker A's old Δ-neighborhood and receiving brokers discard any soft or candidate links that they may have that bypass Broker A and refrain from creating new ones until the move completes. This step completes once A receives a confirmation message from N indicating completion of propagation of m_{init} in A's Δ-neighborhood.

Update phase: Broker A issues a move *in-progress message*, m_{inprog}, at B that includes information about its old and new MOMs as well as the moving path $\mathcal{P}_{A:N\to B}$. This message is propagated within A's post-move Δ-neighborhood and receiving brokers construct a temporary MOM_{tmp} and MST_{tmp} which includes

the state of their new Δ-neighborhoods and subscription anchors after the move. These brokers use MOM_{tmp} and MST_{tmp} for forwarding of publications from this point on. At the same time, they also refrain from creating any soft or candidate links that bypass Broker A. While the move procedure is in progress, Broker A may receive duplicate publication messages routed via Brokers N and B. As a matter of fact, the rationale behind the restriction of requiring nearby brokers to not bypass Broker A while the move is still in progress is to allow effective duplicate detection and elimination at a single broker (*i.e.*, Broker A). Broker A properly discards the duplicates and forwards the publications it receives for the first time according to its new MOM_{tmp} and MST_{tmp}. Finally, propagation of the m_{inprog} message completes when a corresponding confirmation message arrives at Broker A. At this point, Broker A proceeds to the final phase.

Wrap-up phase: Broker A ends its move by issuing a *move end message*, m_{end}, at Brokers N and B. This message propagates within the Δ-neighborhood of Broker N prior to the move and the Δ-neighborhood of Broker B after the move (*i.e.*, to all brokers affected by the move). Receiving brokers that are no longer in A's Δ-neighborhood, discard portions of their MOMs that are downstream of A (including Broker A itself) and substitute subscription anchors that point to A with a broker located at distance Δ on the moving path. Furthermore, brokers that remain or have entered A's new Δ-neighborhood discard their old MOM and MSTs and replace them with their temporary counterparts constructed earlier as part of the move procedure (*i.e.*, MOM_{tmp} and MST_{tmp}). From this point on, these brokers can create soft or candidate links that bypass Broker A.

7 Evaluation

We carried out large-scale experimental evaluations on the SciNet cluster [22] as well as PlanetLab [23] using **Publiy**, our Java-based open-source pub/sub system [20]. SciNet machines each have eight 2.66 GHz 64-bit Intel Xeon CPU cores with 8 GB of memory and PlanetLab machines are a mix of single, dual, and quad core Intel-family $1.8 - 3.2$ GHz CPUs equipped with $1 - 3$ GB of memory. We used several network configurations and pub/sub workload datasets to compare our proposed forwarding strategies against S0, as baseline. Our experimental setups are designed with the anticipated use cases of large-scale pub/sub systems in datacenter or wide-area environments in mind. They are varied in terms of network size, subscription matching distribution, and system parameters such as *fanout*, and bandwidth capacity. Table 1 summarizes different network configurations used in this section. Henceforth, we use the short names, *e.g.*, C1, C2, etc., to refer to the network configuration setups in Table 1.

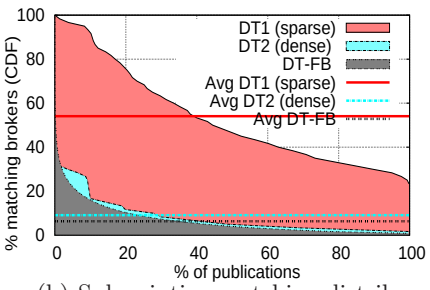
Config.	Net. size	Broker degree	Platform
C1	120	3	SciNet
C2	250	3	SciNet
C3	500	3	SciNet
CPL	21	3	Planetlab

Table 1: Experimental configurations.

In the beginning of each experimental run, brokers propagate subscriptions in the network. In principle, each subscription can be originated from a separate subscriber process. However, due to scarcity of our resources running thousands of clients was infeasible. Instead, we skipped last mile message delivery to clients

Dataset	Config	Subs count	Matching subs per pub
DT1	C1	6,552	8% (Sparse)
DT2		19,472	53% (Dense)
DT3	C2	28,690	9% (Sparse)
DT4	CPL	900	9% (Sparse)
DT5		6,000	54% (Dense)
DT-C	C3	10,000	4% (Covering)
DT-FB	C1	120,000	6.5% (Sparse)

(a) Datasets summary.



(b) Subscription matching distribution for DT1, DT2, and DT-FB.

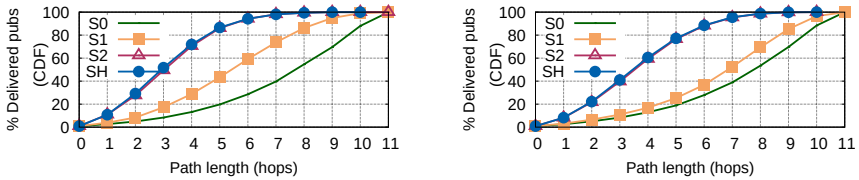
Fig. 8: Workload specification.

and only considered publication forwarding within the broker overlay network. We thus had brokers to locally log delivered publications instead of sending them to an actual subscriber (each broker runs on a dedicated CPU core). We believe that this approach is fair for our comparative study, since direct delivery to subscribers incurs the same amount of processing and bandwidth in all strategies.

Figure 8(a) summarizes the matching distribution of the subscription and publication workload datasets used in our experiments. These datasets are either based on real-world traces of user interactions in social networks [24] (*i.e.*, dataset DT-FB) or are synthesized using a Zipf distribution. This choice was motivated by the study of Liu *et al.* who showed that the popularity of RSS feeds in real-world application scenarios follows Zipf distributions [25]. Figure 8(b) compares the matching distribution of three of our synthesized datasets by illustrating what percentage of publications matches what percentage of subscriptions in the system. As such, datasets with fewer or larger average number of matching subscriptions per publication are categorized as *sparse* or *dense*, respectively. Finally, dataset DT-C has subscriptions that also have covering relationships.

In all the experimental runs in this section, we set Δ to 4. Although a larger value was in principle possible, it complicates the move procedure (which, as stated in Section 6, is complementary to our approach) by increasing the number of affected brokers within Δ-neighborhood of a moving broker. A smaller Δ, on the other hand, limits the range of *fanout* values that we can experiment with. This is due to the fact that brokers only connect to neighbors that are Δ hops away and the size of Δ-neighborhoods indirectly constrains the maximum *fanout*.

Publication forwarding path length: Publication hop count provides valuable insight into the internal workings of the system in each strategy and has a direct impact on delivery delay, throughput and ultimately system scalability. It is expected that if brokers maintain a larger number of soft links the overlay mesh becomes more connected, offering more optimized forwarding paths in the network. Brokers can selectively pick the best forwarding route based on the strategies used in order to reduce the number of pure forwarders. We compared publication hop counts incurred using different strategies experimentally. The results are illustrated in Figure 9 which plots the cumulative distribution function (CDF) of publication propagation path lengths for executions of con-



(a) DT1 (sparse): 348 thousand deliveries. (b) DT2 (dense): 1.03 million deliveries.

Fig. 9: Publication propagation path lengths using configuration C1 and *fanout* of 35.

figuration C1 with datasets DT1 (sparse) and DT2 (dense). The Δ and broker *fanout* parameters in all executions were set to 4 and 35, respectively. Measurements were carried out within a 10 min interval in which exactly the same number of publications are injected in the system at a low rate of $3,600$ msg/sec. At this rate, *no network hotspots* are formed and *all strategies deliver the same number of publications*. This allows us to compare different strategies based on the publication hop count metric independently of other interfering factors.

As shown in Figure 9, compared to S0, strategies S1 and S2 substantially lower publication hop count (data points in the graphs are shifted left). Fewer hops also imply that publications are matched against subscriptions fewer times. Furthermore, as there are no overloaded brokers, SH performs similar to S2. An interesting effect to observe in the graphs is that the difference between S0 and S1 is smaller in the dense dataset compared to the sparse dataset. This is due to the fact that publications in the dense dataset match more subscriptions and the intersection of primary paths (as computed in S1) is likely to bypass fewer neighbors. This brings the performance of S1 closer to S0.

Number of pure forwarding brokers: As mentioned earlier, due to the selectivity of content-based forwarding, some brokers 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 reduce such occurrences by tailoring the actual propagation path of publications based on the relative location of matching subscribers at runtime. Our measurements are reported in Table 2 and show that compared to S0, our forwarding strategies cut the number of pure forwarders by up to 70%. Furthermore, if we consider the *yield* of a pub/sub system as the total number of publications delivered (i.e., arrive at brokers with local matching subscribers) over the total number of pub-

(a) C1/DT1: $348,000$ pubs delivered.

Strategy	Number of pure forwarders	System yield
S0	$559,000$	38%
S1	$348,000$	50%
S2	$216,000$	61%
SH	$195,000$	64%

(b) C1/DT2: $1,034,000$ pubs delivered.

Strategy	Number of pure forwarders	System yield
S0	$1,010,000$	50%
S1	$687,000$	60%
S2	$325,000$	76%
SH	$300,000$	77%

Table 2: Pure forwarding and system yield for different strategies (Configuration C1).

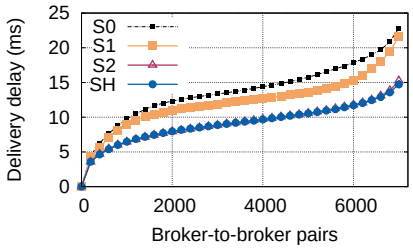


Fig. 10: Publication delivery delay: x-axis shows pairs of source-destination brokers; y-axis is avg. delay for corresponding pair (Conf. C1/DT2, $\Delta = 4$, *fanout* = 35).

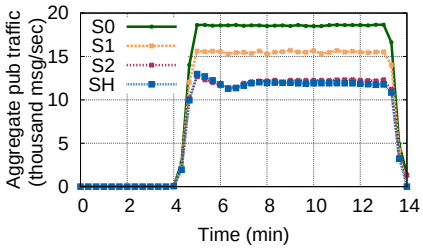


Fig. 11: Aggregate network traffic using different strategies (Conf. C1/DT2, $\Delta = 4$, *fanout* = 35). During first 4 mins subscriptions are propagated (traffic not shown).

lications sent between brokers (including those relayed by pure forwarders), we see that S2 achieves a yield of up to 77%. This is an indication that the system operates more efficiently and better utilizes its resources.

Publication delivery delay: Figure 10 plots average publication delivery delay (vertical axis) for pairs of brokers that host subscribers and publishers (horizontal axis).[4] The overall average delay for S0 (baseline) is 14.0 ms. This is lowered to 12.5 ms for S1 and 9.3 ms for S2 and SH (up to 50% improvement over the baseline). Also, note that this improvement would have been even greater if the input publication traffic was higher and caused the network to congest.

Network traffic: Fewer pure forwarders and higher yield implies that the system can deliver the same number of publications by sending fewer messages among brokers. This lowers network traffic and improves efficiency of bandwidth utilization. Using the same configurations as before, Figure 11 illustrates the aggregate network traffic in terms of the number of publications transmitted. The average network traffic in strategies S0, S1, S2 and SH is $3,400$ msg/sec, $2,850$ msg/sec, $2,200$ msg/sec and $2,200$ msg/sec, respectively. Compared to the baseline, this represents up to 35% improvement in bandwidth utilization. Furthermore, the traffic resulting from load exchange messages (in S1, S2 and SH) incurs less than 2% of the total bytes sent and received. This implies that the bandwidth conservation achieved in our strategies still outperforms its overheads.

Computational overhead: Strategies S1, S2 and SH update WOM and WST after changes to broker links. Our measurements indicate that each update to WOM takes about 0.8 ms. This has a negligible amortized overhead considering the fact that this update takes place roughly every 20 seconds in our implementation. 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 (*i.e.*, DT1). Finally, thanks to our efficient MOM data structure and the use of bit-vectors for path computation the time that it takes for S1 and S2 to forward publications remains unchanged. Detailed measurements regarding computational and memory costs are available in [26].

[4] SciNet uses infiniband interconnect with ultra low latency. We have therefore injected a delay of 1 ms for broker-to-broker communication to account for networking delay.

Impact of *fanout* on broker performance: Increasing *fanout* improves overlay connectivity but comes at the cost of maintaining a larger number of concurrent network connections. This incurs an overhead related to buffer management and TCP's congestion control mechanism. Additionally, as we investigate experimentally in this section, a larger *fanout* limits the advantages brought about by subscription covering techniques and contributes to a degradation in matching performance. This effect is due to *fragmentation* of the subscription space. To clarify this point consider the following simple example: If subscription s_1 covers s_2 and s_2 covers s_3, a broker that possesses only one link ($fanout = 1$) computes $\{s_1\}$ as the covering set. Therefore, publications are matched against *one subscription only*. On the other hand, if the broker possesses two or more links, the covering sets are likely to grow larger making matching more expensive. 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 the broker computes two covering sets each with one subscription, *i.e.*, $\{s_1\}$ and $\{s_2\}$. As a result, matching becomes more time consuming. The exact size of the covering sets, of course, depends on subscription predicates and relative location of issuing subscribers in the network.

We investigated this phenomenon using configuration C3 with 500 brokers and dataset DT-C with covering relationships. We measured publication matching performance in a system using different *fanout* values. Figure 12 illustrates the results normalized based on smallest *fanout* value of 5. The checkered bars represent average size of covering sets over a 120s interval. It is evident that the fragmentation caused by larger *fanout* values increases the size of covering sets. Furthermore, larger covering sets translate to an even sharper increase in predicate evaluation operations needed to match each publication. For example, between *fanout* of 5 and 10 there is a 12% jump in covering set size and a 34% jump in predicate evaluations. This discrepancy 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.

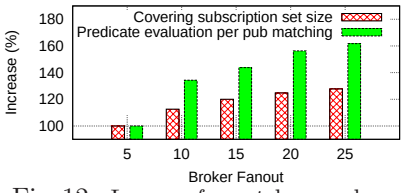
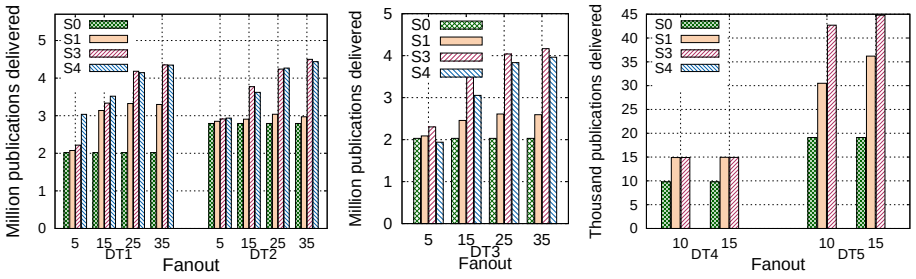


Fig. 12: Larger *fanout* lowers benefits of covering techniques (data normalized based on $fanout = 5$).

System throughput: In large-scale messaging systems that serve thousands of clients in a datacenter or an enterprise, throughput is perhaps the most important aspect of the system. The reduction in the number of pure forwarders in our strategies frees up brokers' valuable bandwidth and processing resources. These resources can be put to use for disseminating a larger number of publications, therefore improving throughput. To study this effect, we carried out extensive experimental analysis using different configurations and datasets on SciNet [22] and PlanetLab [23]. Figure 13(a) illustrates the results using configuration C1 and datasets DT1 (left) and DT2 (right). We used a high aggregate publishing rate of 72,000 msg/min to push the system to the edge. At this rate, the number of deliveries within our 10 min measurement interval gives a clear comparative understanding of the system throughput in each strategy (*i.e.*, some



(a) Configuration C1 (b) Configuration C2 (c) Configuration CPL

Fig. 13: Publications delivered within a fixed measurement interval.

strategies deliver more publications). Two trends are visible in the figure. First, increasing *fanout* from 5 to 35 significantly improves the number of publication deliveries. Second, our forwarding strategies outperform S0 (the baseline) by up to 115%. Both trends are also present in Figure 13(b) in which configuration C2 and dataset DT3 were used. An interesting observation in both figures is that S2 tends to marginally outperform SH. This is indeed expected since at our very high publishing rate many brokers become overloaded and adaptively switch to S1. Although this strategy reduces the load on such overloaded brokers but produces more publications in the network as a whole (compare number of publications sent in Figure 2(b) and Figure 2(c)). This excess traffic degrades the performance of the system as a whole.

In practice, however, brokers are generally provisioned to operate within a safe buffer from their full capacity. Despite this, broker heterogeneity or traffic imbalances may develop occasional network hotspots. In these scenarios, the adaptive nature of SH is useful to prevent formation of network bottlenecks. To investigate this effect, we re-ran configuration C1/DT1 with a low publishing rate of $7,200$ msg/min but throttled brokers' uplinks to be capped at 150 msg/sec. At this rate, some overlay hotspots are formed and the ability of overloaded brokers to dynamically switch to S1 prevents excessive use of their scarce bandwidth. Figure 14 illustrates the throughput and average publication hop count in this scenario using different strategies. It can be seen that SH outperforms S2 in terms of the number of deliveries as it is less likely to develop hotspots.

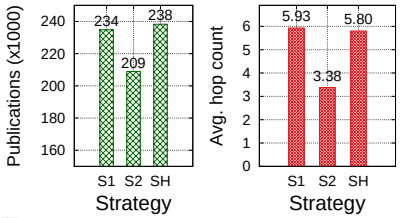


Fig. 14: Performance of different strategies (brokers uplink capped at 150 msg/sec).

PlanetLab results: We verified our results on the shared PlanetLab environment where nodes' CPU and bandwidth capacity is limited and variable. Figure 13(c) illustrates the results of our throughput analysis with configuration CPL and datasets DT4 (left graph) and DT5 (right graph). The measurement interval is 5 minutes, aggregate publishing rate is $3,300$ msg/min and the brokers' uplinks are capped at 10 KB/s. At this rate, the system using DT4 and S0 becomes congested and delivers fewer messages than expected. However, S1 and S2

both reach full delivery goals. Similarly, using DT5, S0 delivers the least number of publications while S1 and S2 achieve much higher (but not full) deliveries.

Facebook dataset: We verified our results using dataset DT-FB which was extracted from real-world traces of user interaction in online social networks [24] (see [26] for methodology and details). In a deployment using configuration C1 with aggregate publishing rate of $3,600$ msg/min and *fanout* of 20, our measurements indicate that S1 and S2 outperform S0 by 37% and 115%, respectively.

8 Related Work

Our approach is related to Resilient Overlay Networks (RON) [27] which uses mesh-based overlay routing to deal with failures. However, unlike RON which targets generic unicast routing, our techniques are specifically tailored for selective publication multicast in content-based pub/sub systems. Furthermore, in contrast to RON which maintains a full-mesh, the use of Δ-neighborhoods in our approach, improves scalability and keeps the overhead to a negligible limit.

Snoeren *et al.* forward publications over *multiple* disjoint paths in a mesh network [15]. Instead of improving bandwidth efficiency, their scheme is concerned with exploiting path redundancy to guarantee message delivery. Overlay reconfiguration techniques adapt the broker overlay by creating links between brokers with similar subscriptions [17,18]. We see these techniques as complementary to our approach. However, as noted in Section 6, changes to the primary network require costly coordinated updates to routing tables of many brokers. Our use of soft links to adapt the overlay avoids the high cost of such full reconfigurations.

In MEDYM [28], the first broker computes a dissemination tree that spans only 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 routing information in publications incurs high overhead, especially for messages destined to a large number of brokers. Li *et al.* [29] 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 nondeterministic propagation patterns in the overlay. In contrast, our approach actively creates soft links to maintain redundant forwarding paths after taking broker load and link quality into account.

9 Conclusions

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

References

1. Global Data Synchronization Network (GDSN): http://www.gs1.org.
2. Cooper, B.F., *et al.*: Pnuts: Yahoo!'s hosted data serving platform. PVLDB (2008)
3. Google Publish/Subscribe (GooPS): CANOE summer school (2009).
4. Li, G., Muthusamy, V., Jacobsen, H.A.: A distributed service-oriented architecture for business process execution. TWEB **4**(1) (2010)
5. Adya, A., Dunagan, J., Wolman, A.: Centrifuge: integrated lease management and partitioning for cloud services. In: NSDI. (2010)
6. Oki, B., Pflügl, M., Siegel, A., Skeen, D.: The informationbus – an architecture for extensible distributed systems. In: SOSP. (1993) 58–68
7. Sadoghi, M., *et al.*: Efficient event processing through reconfigurable hardware for algorithmic trading. Volume 3., VLDB Endowment (2010) 1525–1528
8. PubSubHubBub. http://code.google.com/p/pubsubhubbub/
9. Publish Subscribe Internet Routing Paradigm (PSIRP): http://www.psirp.org.
10. Fidler, E., Jacobsen, H.A., Li, G., Mankovski, S.: The PADRES distributed publish/subscribe system. ICFI (2005)
11. Bhola, S., Strom, R.E., Bagchi, S., Zhao, Y., Auerbach, J.S.: Exactly-once delivery in a content-based publish-subscribe system. In: DSN. (2002)
12. Papaemmanouil, O., Cetintemel, U.: SemCast: Semantic multicast for content-based data dissemination. In: ICDE. (2005)
13. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM TOCS (2001)
14. Cugola, G., *et al.*: The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. Trans on Software Eng. (2001)
15. Snoeren, A.C., Conley, K., Gifford, D.K.: Mesh-based content routing using XML. In: SOSP. (2001)
16. Chand, R., Felber, P.: XNET: a reliable content-based publish/subscribe system. In: SRDS. (2004)
17. Baldoni, R., *et al.*: Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. Computer Journal **50**(4) (2007) 444–459
18. Migliavacca, M., Cugola, G.: Adapting publish-subscribe routing to traffic demands. In: DEBS. (2007)
19. Cheung, A.K.Y., Jacobsen, H.A.: Dynamic load balancing in distributed content-based publish/subscribe. In: Middleware. (2006)
20. Publiy project: http://msrg.utoronto.ca/~reza/.
21. Kazemzadeh, R.S., Jacobsen, H.A.: Reliable and highly available distributed publish/subscribe service. In: SRDS, IEEE (2009) 41–50
22. University of Toronto SciNet Consortium: http://www.scinet.utoronto.ca.
23. PlanetLab testbed: http://www.planet-lab.org/.
24. Wilson, C., Boe, B., Sala, A., Puttaswamy, K.P.N., Zhao, B.Y.: User interactions in social networks and their implications. In: EuroSys. (2009)
25. Liu, H., Ramasubramanian, V., Sirer, E.G.: Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. In: IMC. (2005)
26. Kazemzadeh, R.S., Jacobsen, H.A.: Adaptive multi-path forwarding in the *Publiy* distributed publish/subscribe systems (2011) Tech. rep.: http://msrg.org.
27. Andersen, D.G., Balakrishnan, H., Kaashoek, M.F., Morris, R.: Resilient overlay networks. Computer Communication Review **32**(1) (2002)
28. Cao, F., Singh, J.: MEDYM: Match-early and dynamic multicast for content-based publish-subscribe service networks. ICDCSW (2005)
29. Li, G., Muthusamy, V., Jacobsen, H.A.: Adaptive content-based routing in general overlay topologies. In: Middleware, Springer (2008)