Sub-2-Sub: Self-Organizing Content-Based Publish Subscribe for Dynamic Large Scale Collaborative Networks

Spyros Voulgaris [*†], Etienne Rivière[*‡], Anne-Marie Kermarrec [‡] and Maarten van Steen [†]
[†] Vrije Universiteit, Amsterdam, The Netherlands {spyros,steen}@cs.vu.nl
[‡] IRISA/INRIA, Rennes, France {eriviere,akermarr}@irisa.fr

ABSTRACT

In this paper, we address the problem of constructing scalable content-based publish/subscribe systems. Publish/subscribe systems are asynchronous event-notification systems in which a published event is forwarded to exactly those nodes that have previously subscribed for that event. Subscriptions can range from a simple specification of merely the type of an event to a specification of the value ranges that an event's attributes can have. Notably the latter poses potential scalability problems.

Structured peer-to-peer systems can provide scalable solutions to publish/subscribe systems with simple subscription patterns. For complex subscription types their applicability is less obvious. In this paper, we present SUB-2-SUB, a collaborative self-organizing publish/subscribe system deploying an unstructured overlay network. SUB-2-SUB relies on an epidemic-based algorithm in which peers continuously exchange subscription information to get clustered to similar peers. In contrast to many existing approaches, SUB-2-SUB supports both value-based and interval-based subscriptions. Simulations of SUB-2-SUB on synthetic and reusable workloads convey its good properties in terms of routing efficiency, fairness, accuracy and efficiency.

1. MOTIVATION

Publish/subscribe is an appealing paradigm for distributed and selective content delivery systems. In Publish/subscribe, *subscribers* express their interest in data by registering *subscriptions* with the system, in order to be notified of any forthcoming *events* (issued by *publishers*) matching their subscription. In topic-based publish/subscribe systems, events and subscriptions are associated with a topic name. In content-based publish/subscribe systems, events and subscriptions are represented by arbitrary predicates on attributes. A subscription either specifies an *exact* value ($a = 2$), or covers a value *range* ($a \in [2,7]$).

Peer-to-peer (P2P) systems have been identified as the key to scalability and their self-organizing properties make them natural candidates for large-scale publish/subscribe systems design. Several efficient implementations of P2P topic-based publish/subscribe systems have been proposed [3, 2]. Unfortunately, it is not obvious how to devise a scalable P2P solution for content-based publish/subscribe systems.

Structured P2P overlays [8, 7, 9] have often been favored over unstructured ones to implement content-based publish/subscribe systems. The idea is to map the attribute space of the latter to the identifier space of the former. At one extreme, each attribute is associated with one specific peer. Although this provides efficient routing to interested subscribers, peers hosting popular attributes are quickly overloaded. At the other end of the spectrum, in an attempt to discretize the ranges of attributes, a peer is made responsible for a specific *(attribute, value)* pair. In this case, attaining a scalable implementation for range subscriptions becomes problematic.

In this paper, we step away from structured overlays and propose a fully decentralized and self-organizing approach based on unstructured overlays to deal efficiently with both exact and range subscriptions. Key to our approach, which is called SUB-2-SUB, is that subscribers to the same events are automatically clustered. SUB-2-SUB leverages the overlapping intervals of range subscriptions and creates an unstructured overlay reflecting the structure of the attribute space *and* that of the set of subscriptions. Once subscriptions are clustered, events are directly posted to the proper cluster where they are efficiently disseminated.

A key issue is that SUB-2-SUB is highly reactive to changes in the set of subscriptions. To this end, it deploys an epidemic algorithm to continuously cluster subscribers. Epidemic protocols have proved to converge quickly and to produce failure-resistant overlays. Each peer knows about a few other peers, comprising its *view*. Periodically, a peer exchanges its view with a selected peer, and subsequently each of them updates its view.

In SUB-2-SUB updating the view is based on a proximity metric in the attribute space. In the resulting overlay, subscribers are therefore clustered according to the similarity in their subscriptions. A similar process is followed to navigate publishers to clusters of matching subscriptions. Publishers progress greedily across the network according to the same algorithm and proximity metric, eventually reaching the cluster that contains *exactly* the subscribers which the event should be delivered to. Moreover, within such a cluster, subscribers are loosely organized into a distributed data structure that enables efficient event dissemination.

The rest of the paper is organized as follows. In Section 2 we present the system model, followed by the principles underlying SUB-2-SUB in Section 3. The core epidemic algorithm to build a publish/subscribe overlay is presented in 4. Section 5 presents the simulation results of the system against synthetic and reusable workloads. We then present some related work and conclude.

2. SYSTEM MODEL

A multitude of publish/subscribe systems have been proposed in the literature. Not all of them, however, refer to

[*]Spyros Voulgaris' and Etienne Rivière's research in this project has been partially funded by the Van Gogh grant for European collaborations from Égide.

[†]Spyros Voulgaris is supported by the Onassis Public Benefit Foundation.

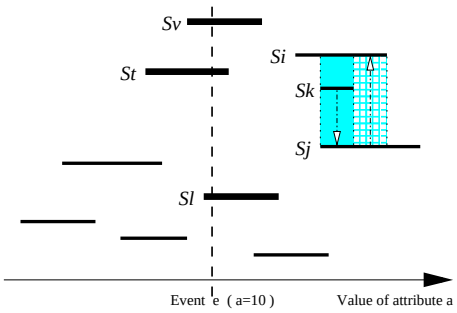

Figure 1: A set of subscriptions and an event.

the same problem. In this paper, we consider content-based publish/subscribe systems where subscribers express their interests through predicates over attributes. In our model, events and subscriptions are associated with one or more attributes and their corresponding values. Subscribers define their interests by means of desired attribute values. A subscription associates with each attribute either a discrete value or an interval. We call the former, *discrete* and the latter *range* subscriptions in the rest of this paper. We consider a *conjunctive* (*AND-based*) subscription model in which *all* of the attributes' conditions should be met for an event to match a subscription.

The attribute space is composed of N floating-point attributes and supports subscriptions on both *discrete* attribute values and *ranges*. More formally, we assume a fixed number, N, of attributes, $A_1, \ldots, A_N$, with values in $\mathbb{R}$ (the set of real numbers). Attributes can alternatively be assigned values of any type that can be directly mapped to $\mathbb{R}$, such as integers, enumerations, boolean values, or character arrays.

Subscriptions are conjunctions of predicates on one or more attributes. A predicate can denote either an exact value (e.g., $A_i = v$), or a continuous range of values (e.g., $A_i \in [v_{min}, v_{max}]$). A subscription can have at most one predicate per attribute. Multiple exact values or multiple non-continuous subranges on a single attribute can be modeled as multiple separate subscriptions. Attributes not referred to in a subscription (wildcards) are assumed to cover the whole attribute space, that is, their value is indifferent to the subscriber. An example subscription for a 5-attribute system is $\langle A_2 = 30 \ \wedge \ A_5 \in [2.2, 2.7] \rangle$.

Events are N-sized vectors specifying exact values for *all* attributes. An example event for a 5-attribute system is $\langle \{A_1, A_2, A_3, A_4, A_5\} = \{5, 30, -2.5, 20, 1.87\} \rangle$.

In the remainder of this paper, we will consider range subscriptions, i.e., subscriptions composed of a set of predicates, each specifying a range of values. Exact-value predicates are considered as a special, and simpler, case of a range subscription.

3. SUB-2-SUB IN A NUTSHELL

SUB-2-SUB is an autonomous, self-organizing P2P event-notification system that supports multi-attribute subscriptions. *Autonomous* implies that the dissemination of events to all interested nodes is accomplished by the cooperation of interested nodes themselves, eliminating any dependency on relay servers or dedicated elements. *Self-organizing* refers to the fact that nodes organize themselves in a structure that enables their cooperation for event dissemination in a completely decentralized manner. The self-organizing property of SUB-2-SUB relies on the use of an epidemic algorithm to cluster similar peers. Its efficiency relies on the fact that overlapping subscriptions are leveraged so that *(i)* only interested subscribers are reached by an event and *(ii)* subscribers do not miss any event matching their subscription.

Epidemic-based clustering. SUB-2-SUB forms an unstructured overlay network in which each peer is associated with a single subscription. Multiple subscriptions are handled by running multiple virtual peers on a single physical node, each virtual peer maintaining its own set of links. SUB-2-SUB implements an epidemic algorithm to automatically cluster similar subscriptions. Periodically, peers exchange information to discover similar peers to form clusters with. Note that the resulting clusters do not have explicit boundaries. Figure 1 depicts an example of a set of subscriptions for a single attribute scheme. Each line represents a range subscription. The epidemic algorithm ensures that peers are automatically clustered so that when an event specifying a value for attribute a (e.g., $e : \langle a = 10 \rangle$ in Figure 1) is published, all interested subscribers (S_v, S_l and S_t in Figure 1) get it.

Organizing range subscriptions. A key observation underlying SUB-2-SUB's design is that every subscription by peer i essentially specifies an N-dimensional subspace $S_i \subset \mathbb{R}^N$, which we refer to as a *hyperspace*. As a consequence, we are interested only in those events that fall into $\mathcal{S} = \bigcup S_i$. The principle behind SUB-2-SUB is that we automatically partition $\mathcal{S}$ into M disjoint hyperspaces $\mathcal{S}_1, \ldots, \mathcal{S}_M$ such that

$$\forall \ 1 \leq m \leq M \ : \ [\mathcal{S}_m \cap S_i \neq \emptyset] \Rightarrow [\mathcal{S}_m \subseteq S_i]$$

Furthermore, we demand that M is minimal: there is no partitioning with fewer parts that can satisfy this constraint.

To this end, we let peers periodically exchange their subscriptions. If two peers i and j note that $S_{ij} \equiv S_i \cap S_j \neq \emptyset$, they will record this fact and maintain references to each other (how this is done is described below). For example in Figure 1, S_i and S_j satisfy the above condition for a given range and get connected. When discovering a third peer k with $S_{ijk} \equiv S_i \cap S_j \cap S_k \neq \emptyset$, peers i, j and k will further organize into a structure associated with S_{ijk} such that an event $e \in S_{ijk}$ will be efficiently disseminated to the three peers. Both i and j will still maintain references to each other, but now for the subspace $S_{ij} - S_{ijk}$. Figure 1 illustrates this process: when S_k joins the network it gets connected to S_j for the shaded range while S_j and S_i remain connected for the hatched range.

A publisher of an event e joins the overlay identically to subscribers, and will eventually find the set $\mathcal{S}_m$ which e belongs to. At that point, e is disseminated to the members associated with $\mathcal{S}_m$. Note that, provided $\mathcal{S}$ is indeed partitioned along the lines we just described, e will reach *only* nodes that are interested in it, and no other ones.

This way of matching a publisher to the relevant subscribers ensures that an event will be delivered *only* to subscribers interested in it, but not necessarily to *all* of them. To achieve dissemination to all interested subscribers, we further organize nodes associated with $\mathcal{S}_m$ in a ring, as we will see in the following section.

4. EPIDEMIC-BASED PUB/SUB

Let $\mathcal{N}_m$ denote the set of peers associated with $\mathcal{S}_m$. The

major issue that SUB-2-SUB needs to solve is to ensure that each peer i is contained in exactly those sets $\mathcal{N}_m$ for which $\mathcal{S}_m$ intersects with S_i: $[i \in \mathcal{N}_m] \Leftrightarrow [\mathcal{S}_m \cap S_i \neq \emptyset]$. To this end, we let each peer i maintain a reference to another node j if S_i and S_j intersect, and such that this intersection is not yet fully covered by the subscription of another node to which i has a reference. Initially, i's goal is to make sure that its entire subscription S_i is covered (unless there are parts for which it is truly the only subscriber). The principal idea is that when an event $e \in S_i \cap S_j$ is published, either i passes the event on to j or *vice versa*. As a consequence, we also need to ensure that *all* peers interested in e are one way or another directly or indirectly linked to each other. To this end, those peers are organized into a ring-like structure, described below.

4.1 Building the overlay

We let nodes self-organize into bidirectional rings that represent the sets $\mathcal{N}_m$ mentioned above. Each node is equipped with a random *sequence ID* uniformly drawn from a large identifier space. If node i discovers that node j covers part of its subscription, it will keep a reference to j. However, as soon as another node k is discovered that covers the same or larger area, but with a sequence ID that lies between that of i and j (using cyclic arithmetic), i will trade the reference to j for that of k. The use of (random) identifiers allows for a deterministic organization of peers so that the event-spreading algorithm is sure to reach all interested subscribers.

Of course, maintaining a ring allows only for a linear dissemination speed. To improve on this, each node also maintains links to randomly chosen peers. When an event $e \in \mathcal{S}_m$ needs to be disseminated, not only is it sent along the ring for $\mathcal{N}_m$, it is also sent to randomly chosen peers that have interest in e as well. Effectively, this short-cuts the ring. By its recursive nature, this forwarding requires only a logarithmic-like number of steps.

Finally, to actually discover nodes, we deploy CYCLON [11], an epidemic protocol by which a node maintains a list of randomly selected neighbors (its view), and regularly exchanges views with other peers. This also serves as a membership management protocol. Nodes join by contacting any one node of the network, and CYCLON takes care of integrating them in the system.

These observations lead to three different types of links. *Random links*, i.e., links to randomly selected peers in the overlay, are needed to discover nodes, and to keep the whole overlay connected in a single partition. *Overlapping-interest links* reflect the similarities between subscriptions and are used to send published events to random other interested peers (and to speed up event dissemination). Finally, *ring links* are used to build a ring of nodes for each set $\mathcal{N}_m$. Figure 2 depicts these three types of links, for a set of subscriptions over one attribute.

For each type of link, a peer maintains a separate view with each associated protocol.

Random links. Several approaches may be used to randomly sample peers in an unstructured peer-to-peer overlay [6]. In SUB-2-SUB we use CYCLON [11], an epidemic protocol that has shown to produce overlays that strongly resemble random graphs [6].

Overlapping-interest links. Such links are maintained using a proximity-based epidemic protocol, here we use VICINITY [12]. The basic idea of proximity-based epidemic protocols is that peers, upon epidemic view exchanges, keep links to the closest nodes according to a given proximity metric (here proximity refers to a distance in the attribute space). In SUB-2-SUB proximity is defined as 0 if two nodes have overlapping interests and otherwise as the Euclidean distance in the respective hyperspaces that represent two nodes' subscriptions. Formally, if $S_i = [l_1^i, r_1^i] \times \cdots \times [l_N^i, r_N^i]$, then

$$d(i,j) = \sqrt{\sum_{k=1}^{N} \left(min\{r_k^i, r_k^j\} - max\{l_k^i, l_k^j\}\right)^2}$$

for peers i and j with non-overlapping subscriptions S_i and S_j. In this way, each node builds an ordered list of nodes with similar interests.

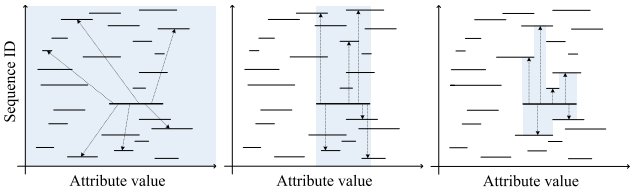

Figure 2: Each subscriber maintains three sets of links. From left to right : random links, overlapping interests links, and ring links. Shaded areas denote where links of the respective type are appropriate for this subscriber.

```
function select peers for node j
    var space: HyperSpace
    var nodes, selected: set of Node init ∅
    nodes ← l1.view + l2.view + l3.view
    for direction in {ascending,descending}
        space ← j.space
        foreach N ∈ nodes from j.id by direction
            if N.space intersects space then
                space ← space − N.space
                selected ← selected + {N}
            end if
        end foreach
    end for
```

Figure 3: Pseudocode for selecting peers.

Ring links. To maintain such links, periodically each peer i initiates a view exchange with subscriber j, selected among i's ring neighbors. In this case, i merges all its views (i.e., including the ones for the random and overlapping-interest links) into a single container. It subsequently goes through the subscribers in this container in increasing sequence ID order (and cycles when reaching the maximal sequence ID) and selects a subscriber only if it intersects j's interest space at some region not yet covered by already selected subscribers. This process is then repeated, but now iterating in decreasing sequence ID order. Note that in this way, we build a bidirectional ring. The selected subscribers are then sent to j, which subsequently performs the same logic. The pseudocode for selecting ring links is shown in Figure 3.

Note that ring links are indifferent to publishers. Indeed, publishers build views for only random and overlapping-interest links, and gossip greedily (as fast as they can) to reach *any* matching subscriber, independently of its sequence

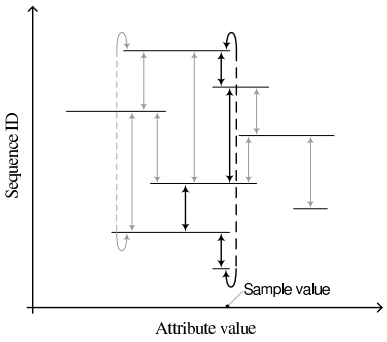

Figure 4: An event dissemination ring highlighted.

ID. As we will see in Section 5, this permits them to find a matching subscriber in a very small number of steps. If more steps than a small threshold elapse, they can safely assume that no subscriber in the whole network is interested in their event(s).

4.2 Spreading events

Given the dissemination overlay described above, publishing events is a simple task. All a publisher has to do is locate *any* one matching subscriber for its potential event(s), and deliver the event(s) to it. From that point on, dissemination is taken care of by the matching subscribers themselves. Figure 4 depicts a set of subcriptions and the associated ring, for a sample event value (only rings links are shown).

In particular, we assume that each node is running a simple *dissemination algorithm* as a daemon thread listening to incoming events and forwarding them accordingly. The daemon thread is activated when a node receives an event. It first looks up the node's recent event history. Previously seen events are ignored. New events are delivered to the application, and subsequently forwarded in two respects. First, an event is forwarded to the node's two adjacent neighbors (if any) along the event's ring. Second, it is forwarded to a small number (typically only one or two) of additional matching subscribers, by following random shortcuts in the event's ring. Obviously, a node does not send an event back along the same link it received it through.

Four facets of the dissemination algorithm are worth noting, namely its behavior with respect to *hit ratio*, *propagation speed*, *spam ratio*, and *load balancing*.

Hit ratio and propagation delay are dealt with by ring links and random shortcuts, respectively. Forwarding along ring links guarantees that events are sequentially propagated to *all* corresponding matching subscribers, achieving a hit ratio of 100%. Random shortcuts to matching subscribers are followed only to boost propagation speed. Indeed, following the ring links alone requires linear time to cover all interested nodes. By each node forwarding incoming events to as few as one random matching node, dissemination completes in close to logarithmic time.

Spam is entirely out of the question in this dissemination algorithm. Clearly, no node forwards an event to another node, unless the latter is interested in that event. The only scenario in which a node might receive spam messages, is if it has recently modified its subscription, and its updated subscription has not yet spread enough. In that case, it may still receive events matching its previous subscription.

Finally, with respect to load balancing, two points are worth emphasizing. First, no dissemination load is imposed on irrelevant subscribers. Second, load is evenly balanced across matching subscribers, as each of them receives an event once or a few more times, and upon first reception forwards it to the same small number of nodes: up to two adjacent neighbors, and a few random ones.

5. EVALUATION

In this section, we evaluate SUB-2-SUB by simulation under synthetic subscription workloads. We focus on four key issues: overlay construction, hit ratio, propagation speed, and complexity for publisher and subscriber joins. Spam ratio is not considered, as it is fundamentally eliminated by our design.

We built and evaluated SUB-2-SUB on *PeerSim*[1], an open source Java simulation framework for P2P protocols.

Experimental setup. In lack of real-world subscription datasets, we generated synthetic ones as follows. N-attribute subscriptions were represented as N ranges in $[0 \ldots 1]$, one for each attribute. A range's center was chosen following the respective attribute's *interest distribution*. A range's width was determined by the attribute's *width distribution*.

In each experiment we applied the same interest distribution to all N attributes: either *uniform* or *power law*. The former represents a natural unbiased workload. The latter, known as *Zipf*, is admitted to be a good approximation of interest popularity and results in subscription sets closer to expected social behavior, exhibiting popular and rare values. It also results in more interesting experiments, as there is higher overlap around the "center" of the interest space, and lower towards its edges, resulting in rings of various lengths. The width distribution was fixed to *power law* centered at 0, with $\alpha = 4$, to account for both wide range and (nearly) exact subscriptions.

In evaluating SUB-2-SUB we considered schemes of up to five attributes. The number of subscribers was fixed to 10,000 for all experiments.

We tested each experiment's effectiveness by observing the dissemination of 10,000 test events. Test events were picked at random, ensuring each one had at least two matching subscribers, to make dissemination meaningful.

Jump-starting SUB-2-SUB. We first test our algorithm's ability to jump-start a SUB-2-SUB overlay from scratch. Nodes started gossiping at the same time, having been initiated with a single random link in their CYCLON views, ensuring the overlay formed a connected graph. We recorded the topology evolution by keeping statistics over the ring links associated with each of the 10,000 test events.

Figure 5 shows the evolution of ring construction per cycle, for four experiments. We can see that after 40 cycles, all rings are fully set up.

Due to limited space, and having seen that rings are constructed in a small number of cycles in all cases, the remaining evaluation concentrates on a single experiment, namely the one with three attributes and power law interest distribution. This experiment is the most interesting for testing event dissemination and propagation speed, as the rings it involves range from very small (2 subscribers) to quite large (246 subscribers). In five-attribute schemes, rings are trivially short (2-3 subscribers) due to the very large subscription

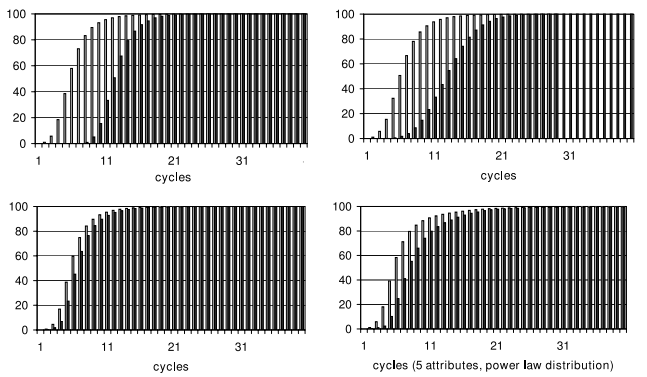

Figure 5: Construction of the rings in time. Light bars show the percentage of ring links already in place. Dark bars show the percentage of rings that are complete. 10K nodes.

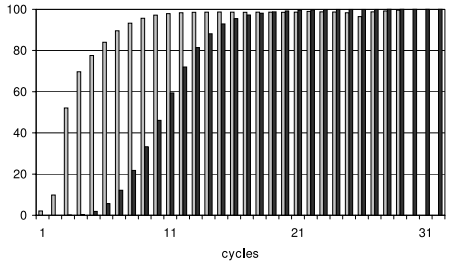

Figure 6: Event dissemination. Light bars show the hit ratio for *non-complete* disseminations. Dark bars show the percentage of disseminations that were complete (events delivered to all their matching subscribers). 10K nodes; 3 attributes; power law interest distribution.

space. (Fig. 7(a)).

Event dissemination. We apply the SUB-2-SUB dissemination algorithm by forwarding an event to *one* random matching subscriber in addition to its two adjacent ones.

Figure 6 presents the performance of SUB-2-SUB with respect to event dissemination. It is worth noting that, by comparison to Fig. 5(upper-right), complete dissemination is achieved even *before* all rings are in place. This comes as a result of (also) forwarding to random overlapping links.

Propagation speed. We now examine the speed, in terms of the number of hops at which events spread. We are specifically interested in the number of hops for *complete dissemination*, that is, the number of hops elapsed from the moment a publisher delivers an event to some matching subscriber, until the event reaches the last one of them.

Figure 7(b) shows the number of dissemination hops as a function of the number of subscribers matching the respective events. Clearly, the number of hops increases with the number of matching subscribers. However, as a result of short-cutting the rings in disseminating events, this relation is of logarithmic fashion.

Single joins. Jump-starting SUB-2-SUB consists a worst case scenario, as the whole overlay starts from a completely

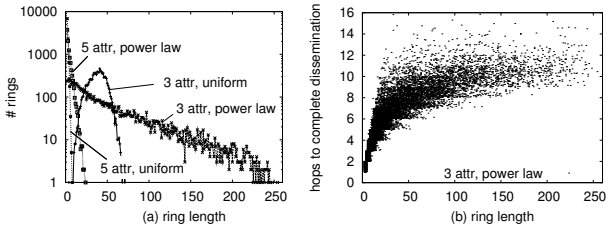

Figure 7: (a) Distribution of ring lengths. (b) Hops to complete event dissemination, as a function of the number of matching subscribers (ring length). 10K nodes; 3 attributes; power law interest distribution.

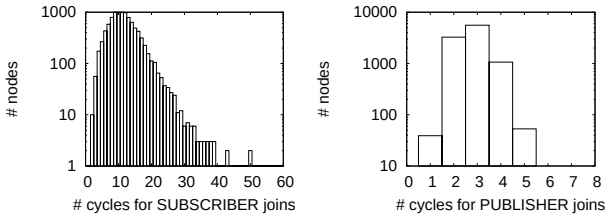

Figure 8: Cycles it takes subscribers and publishers to join.

non-clustered state. We now take a look at the other end of the spectrum, measuring the speed of *single* node joins to an already converged overlay.

Starting from the converged state of our experiments, each subscriber (and links to it) was individually removed to let it join again. The number of cycles it took to re-establish all ring links of the rejoined subscriber to its neighbors and *vice versa*, gives the distribution shown in Figure 8.

For publishers, on the other hand, joining is a simpler task, as they are only interested in reaching *any* matching subscriber, independently of its sequence IDs. Figure 8 also shows the distribution of cycles it takes publishers to join, starting from a random node. It is worth noting that all 10,000 publishers we tested joined in five or less cycles. This is important, as a publisher can safely assume there is no subscriber matching its event(s) after a low threshold of cycles (i.e., in the order of 10 or 20).

6. RELATED WORK AND CONCLUSIONS

Scalability of peer-to-peer systems makes them natural candidates to implement large-scale publish/subscribe systems. In this paper, we presented the design and evaluation of SUB-2-SUB, a scalable, self-organizing peer-to-peer approach for content-based publish/subscribe in collaborative environments. SUB-2-SUB deploys an unstructured overlay where subscribers are clustered in efficient dissemination structures, based on shared interests. To the best of our knowledge, SUB-2-SUB is the first attempt to build publish/subscribe overlays using epidemic-based algorithms, thus exploiting their ability to handle dynamic environments.

Unlike SUB-2-SUB, previous peer-to-peer approaches for content-based publish/subscribe have mainly focused on structured overlays. Among them, Meghdoot [5] uses an extension of the CAN DHT [7]. It maps subscriptions to a $2 \times k$-Euclidean space, where k is the number of attributes. Each attribute is represented by two dimensions, corresponding

to its minimum and maximum allowed values respectively, allowing for range subscriptions. Unlike SUB-2-SUB's autonomous and self-contained operation, Meghdoot employs a separate set of dedicated nodes for storing subscriptions and disseminating events. Meghdoot deals with sparse interest distribution with CAN zone replication, which may be computationally expensive to maintain in highly populated parts of the space. Terpstra et al. [10] proposed to leverage the properties of the Chord DHT [9] to implement an event filtering system. Distributed nodes are dynamic brokers organized in a graph. They use subscription merging and covering to provide scalability, acting similarly to traditional content-based filtering systems based on a dedicated set of brokers. However, such a set of brokers may be hard to maintain efficiently in highly dynamic environments. Finally, Costa et al. proposed to use epidemic-based algorithms to enhance reliability of existing publish/subscribe systems [4].

We conclude that SUB-2-SUB is an appealing alternative to existing solutions for content-based publish/subscribe. It offers a scalable, autonomous, and self-organizing system, combining the resilience of epidemic-based overlays with the expressiveness of the content-based model.

REFERENCES

[1] peersim. http://peersim.sourceforge.net.

[2] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *SIGCOMM '02*, pages 205–217, Pittsburgh, PA, 2002.

[3] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A Large-scale and Decentralized Publish-Subscribe Infrastructure. *IEEE JSAC*, 20(8), October 2002.

[4] Paolo Costa, Matteo Migliavacca, Gian Pietro Picco, and Gianpaolo Cugola. Introducing Reliability in Content-Based Publish-Subscribe through Epidemic Algorithms. In *DEBS*, pages 1–8, San Diego, CA, 2003.

[5] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: Content-Based Publish/Subscribe over P2P Networks. In *Middleware*, pages 254–273, Toronto, Canada, 2004.

[6] Màrk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In *Middleware*, pages 79–98, Toronto, Canada, 2004.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, pages 161–172, San Diego, CA, August 2001.

[8] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware*, Heidelberg, Germany, 2001.

[9] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *ACM/IEEE Trans. Netw.*, 11(1):17–32, February 2003.

[10] Wesley W. Terpstra, Stefan Behnel, Ludger Fiege, Andreas Zeidler, and Alejandro P. Buchmann. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In *DEBS*, pages 1–8, San Diego, CA, 2003.

[11] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. CYCLON: Inexpensive membership management for unstructured P2P overlays. *J. Network Syst. Mgmt.*, 13(2), 2005.

[12] Spyros Voulgaris and Maarten van Steen. Epidemic-style Management of Semantic Overlays for Content-Based Searching. In *EuroPar*, Lisboa, Portugal, September 2005.