Subscription Awareness Meets Rendezvous Routing

Fatemeh Rahimian, Sarunas Girdzijauskas, Amir H. Payberah, Seif Haridi

Swedish Institute of Computer Science
Stockholm, Sweden
email: {fatemeh,sarunas,amir,seif}@sics.se

***Abstract*—Publish/subscribe communication model has become an indispensable part of the Web 2.0 applications, such as social networks and news syndication. Although there exist a few systems that provide a genuinely scalable service for *topic-based* publish/subscribe model, the *content-based* solutions are still suffering from restricted subscription schemes, heavy and unbalanced load on the participating nodes, or excessively high matching complexity. We address these problems by constructing a distributed content-based publish/subscribe system by using only those components that are proven to be scalable and can withstand the workloads of massive sizes. Our publish/subscribe solution, *Vinifera*, requires only a bounded node degree and as we show, through simulations, it scales well to large network sizes and remains efficient under various subscription patterns and loads.**

I. Introduction

The amount of data in the digital world that surrounds us is increasing very rapidly, thus, finding the relevant pieces of information becomes even more challenging. Publish/subscribe systems, which are now pivotal to many Web 2.0 applications, especially On-line Social Networks (OSNs) like Twitter, Facebook and Google+, leverage this problem by providing users with only the information they are actually interested in, e.g., a friend's status update, sport news, or a music playlist. As a result, when some new data is generated, the interested subscribers are notified. All these examples, which classify data into coarse-grained predefined categories or topics, are known as *topic-based* subscriptions. On the other hand, subscriptions that define a more fine-grained filter over the content of the generated data, are called *content-based*. These subscriptions usually consist of continuous ranges of values over several attributes that describe the content. For example, a user may be interested to get notified if the local temperature is below zero between 6am and 6pm.

The traditional model to provide a publish/subscribe service uses a central server or broker to maintain node subscriptions [2][6][14]. The published data is also sent to this central point and is matched against the existing subscriptions. Since subscription maintenance and matching are done centrally in this approach, the server could become a bottleneck as the number of users and subscriptions grow. Consequently, researchers have studied distributed publish/subscribe systems, in particular peer-to-peer (P2P) solutions, as an alternative design paradigm to the centralized model. Currently, a wide range of solutions are proposed for topic-based publish/subscribe over P2P overlays [4][11][16][32]. The topic-based solutions, however, can not be readily reused for the content-based model, due to the conceptual differences, i.e., discrete independent topics versus multiple continuous ranges over various attributes. Hence, a number of solutions have been proposed particularly for P2P content-based publish/subscribe [19][31][36][39], spanning from the designs based on unstructured gossip driven overlays to highly structured overlays with rigid event dissemination structures.

In particular, at one end of the design space we find a family of solutions that are subscription aware [19][36]. These solutions, partition the data space into subspaces that include each and every subscriber that is interested in that subspace. Published data is routed to the subspace that it belongs to, and is delivered to the subscribers in that subspace. As we describe in Section II, these systems perform well under simple workloads, but fail to deliver an efficient service to massive number of users with multi-dimensional subscriptions, mainly because they require unbounded number of connections per node. Moreover, despite having to maintain potentially numerous connections, these solutions can not provide an upper bound guarantee for average delivery path length.

At the other end of the design space lie DHT-based solutions, such as [31][37][39], that exploit a technique, known as *rendezvous routing* [5]. To enable rendezvous routing all the nodes and attributes are embedded in an identifier space, by taking a random identifier. Also, a distance function is introduced to make a greedy routing possible. The node with the closest, but higher, identifier to the attribute identifier is selected as the responsible node, i.e., *the rendezvous node*, for that attribute. Every subscriber node that performs a greedy routing (lookup) for an attribute identifier, therefore, ends up at the rendezvous node for that attribute. Next, the reverse routing path, i.e., from the rendezvous node to the subscriber node, is used for data dissemination. Consequently, the dissemination structure consists of a single tree per attribute, thus, often consisting of a handful of dissemination trees for the whole node population. Note, this is different from the multiple rendezvous based trees for topic-based subscription model (e.g., as in Scribe [10]), because, as opposed to the topic-based model, where a subscriber to a topic wants to receive all the events relevant to that topic, in the content-based model nodes that join an attribute tree, are often subscribed to only a subset of the possible values for that attribute. Hence, while this approach does not require an unbounded node degree, the constructed dissemination trees, which blindly deliver all the events to every node on the tree, are very inefficient.

In fact, the aforementioned state-of-the-art solutions are forced to choose a trade-off between scalability (bounded node degree and efficient routing) and overhead (both in volume and distribution), thus, fail to provide a genuinely distributed publish/subscribe service for Internet-scale applications. Such state of the design space inspired us to work on a solution which would not require unsuitable trade-offs and could retain all the desirable properties of a scalable system under any scenarios. As a result, we propose *Vinifera*, which to the best of our knowledge, is the first P2P content-based system that simultaneously fulfills all the fundamental requirements of a scalable distributed service.

Vinifera is empowered by a gossip-based topology management service and clusters the nodes with similar subscriptions. These clusters are later exploited to create efficient data dissemination structures. The same gossiping process, also, embeds a navigable small-world structure into the overlay after each node is assigned an identifier, selected from a globally known identifier space. This enables a distributed greedy distance minimizing routing algorithm to find short paths between any two nodes, which in turn, allows us to utilize the aforementioned rendezvous routing technique [5]. However, in contrast to other content-based publish/subscribe systems that construct a single delivery tree per attribute, thus, suffer from an unbalanced load and large traffic overhead, Vinifera constructs a forest per attribute, where the roots of the trees in the forest are the rendezvous nodes for the attribute values, thus, the load is distributed over all the participating nodes. These trees are dynamically constructed based on the user subscriptions.

Vinifera forest is constructed by utilizing an order preserving hash function [13], that maps each and every attribute domain to the node identifier space. For example, if an attribute has a domain [a, z], this range is mapped to the whole identifier space (say between 0 and 1) and every node in the overlay takes the responsibility for a part of this range that falls between itself and its predecessor in the identifier space. For example, let us assume nodes X, Y, and Z are responsible for ranges [a, d], [d, f] and [f, g] in the attribute domain, which are in turn, mapped to [0, 0.1], [0.1, 0.3], and [0.3, 0.4] in the identifier space. Then, nodes that subscribe to any value in the first range, route towards node X, while nodes that subscribe to the values in the second or third ranges, route towards nodes Y or Z, respectively (Note, a node can subscribe to a range, which contains multiple rendezvous nodes, for example, a subscription for the range [b, e] will be routed to both nodes X and Y, each being responsible for a part of the subscription). Hence, multiple small trees are constructed for event delivery for this attribute. We further enhance the load balancing in Vinifera, by a novel technique that enables it to deal with non-uniform subscriptions and publications. Thus, we ensure an evenly distributed load, even in case the data in some regions of the identifier space is more popular or is published more frequently than the other regions.

The resulting balanced load in Vinifera is of critical importance, not only because it implies fairness and a higher resource utilization, but also, and most importantly, because it enables the system to function under massive data publications and tolerate failures.

We run extensive simulations to evaluate multiple aspects of the performance, namely scalability, fault tolerance, load balancing and congestion control. We compare Vinifera to a baseline system, constructed based on a state-of-the-art solution, eFerry [39], which is a purely small-world overlay, oblivious to node subscriptions. Section II shows that this baseline solution is also conceptually equivalent to Ferry [39] and CAPS [31]. We show that, compared to the baseline system, Vinifera generates only one third of the traffic overhead, while the load is evenly distributed among the nodes and the delivery paths are up to four times shorter. We also show that Vinifera outperforms the baseline solution in the presence of churn, derived from real-world traces, and under intensive publications.

To summarize, our contribution is a genuinely scalable fault-tolerant multi-dimensional content-based publish/subscribe system with a bounded node degree requirement, a logarithmic worst-case bound on the delivery path length, and small and balanced load on the nodes. We achieve these properties by utilizing (i) an overlay topology that adapts to user subscriptions and exploits the similarity of subscriptions, in order to reduce the amount of traffic overhead that is generated in the network, (ii) constructing multiple efficient dissemination paths, instead of a rigid single tree structure, and (iii) a load balancing mechanism that enables the system to work under massive workloads.

II. Related work

As we briefly discussed in the introduction, there exist a number of solutions for building distributed content-based publish/subscribe systems [19][31][36][39]. In this section, we will have a closer look at these systems.

Meghdoot [19] exploits the idea of mapping each node subscription to a point in a $2d$ dimensional space, where d is the number of attributes/dimensions in the subscription scheme. Then, a CAN [33] overlay is utilized for routing the messages. Although matching events against subscriptions can be nicely done in Meghdoot, the routing is not efficient, due to the inherent inefficiencies in CAN overlay. Moreover, node degree could grow linearly with the number of attributes. The load on the nodes is also very unbalanced, depending on where in the CAN overlay the node is positioned.

Sub-2-Sub [36] takes a completely different approach. It clusters the subscription space into multiple subspaces, where each subspace includes all and only the nodes that are subscribed to the whole subspace. From then on, each subspace is treated like a topic in a topic-based model. A ring is constructed over each subspace for disseminating the events inside that subspace. The problems are two fold: firstly, it is difficult to construct the subspaces, if subscriptions are complex. In Hyper [38], which is a non P2P solution for content-based publish/subscribe, it is proved that solving such a problem is NP-complete. The existence of churn in the P2P networks makes this problem even more challenging.

Secondly, maintaining a ring per subspace implies that if a subscription is split into many subspaces, then the node has to join many overlays at the same time. Therefore, the node degree and maintenance cost could grow very large.

Ferry [39] is yet another approach to enable subscriptions over multiple attributes by employing a structured overlay network. Every node hashes the attribute names and sends its subscription to a rendezvous (RV) node, which is responsible for one of the generated hash values, preferably to the closest one. All the subscriptions are then maintained at the RV nodes. Upon an event publication, the event is delivered to all the RV nodes and will be routed towards the subscribers, accordingly. The strong point in Ferry is that the node degree is bounded regardless of the number of attributes in the subscription scheme. However, since the nodes subscribe for the hash of the attribute names, the routing structure solely depends on the subscription scheme in the system. For example, if there is only one attribute in the model, then one RV node and one delivery tree will exist. Therefore, the load on the nodes will be extremely unbalanced. The RV node not only receives all the published events in the system, but also has to match each and every event against all the existing subscriptions, before relaying the received events. An effort to solve the problems in Ferry is presented in eFerry [37]. The approach is to use different combinations of several attributes, for subscription registration. The proposed mechanisms exhibits desirable load balancing properties only for the publish/subscribe system with extremely large number of attributes, while is still inefficient for the usual systems with one or few attributes.

Another solution, that also requires a bounded node degree, is CAPS [31]. Similar to Ferry, CAPS uses the rendezvous model for subscription installation and event delivery. The main difference is that instead of a single key per attribute, it generates a set of hash values for each subscription, and installs a node subscription in multiple RV nodes in the overlay. The matching is then performed at those RV nodes and events are forwarded along the overlay links from the RV nodes to the subscribers. The problem in CAPS is that a subscription may be translated into too many keys to be installed, and could potentially result in a high traffic in the network. Moreover, the matching is performed centrally at the RV nodes and there is no mechanism for load balancing.

Pyracanthus [1] is another solution, which uses order preserving hashing to enable range queries for content-based publish/subscribe. However, it has a high maintenance cost as it stores a node subscription in all the rendezvous nodes across all the attributes. Moreover, event publication is very costly in Pyracanthus, since the publisher requires to collect all node subscriptions from the rendezvous nodes for all the attributes. It then selects the matching subscribers among the collected subscriptions, and forwards the event to them.

BlueDove [28] is yet another solution for multi-dimensional content-based subscriptions, which is particularly designed for well-engineered environments like clouds. In such environments, data center servers are connected by high speed local networks, packet loss rate is very low, and servers stay on-line for long periods of time.

There are also some related work on enabling range queries over P2P networks [3][8]. Mercury [8], for example, supports multi-dimensional rang-based searches, while it guarantees efficient routing and load balancing. Nevertheless, it does not provide all the necessary propertied of a fully-fledged publish/subscribe system, since it lacks the mechanisms for installing user subscriptions and event delivery. Moreover, the construction of the overlay in Mercury is oblivious to user interests.

Another set of related work is focused on how to filter data content in the overlay networks [12][17][27]. However, these works are orthogonal to our work and can be complementary to Vinifera. In particular, we can utilize [12] on top of our dissemination trees in order to better filter out the published content. The focus in Vinifera is on building a topology that exploits user subscriptions to enable efficient data dissemination structures.

III. Architectural Model

Vinifera is a multi-layer solution, where each lower layer provides a service to its upper layers. The architectural model of Vinifera (Figure 1(a)) consists of three main layers:

Random overlay. On the bottom layer we have a random network, which we construct by a gossip-based peer sampling service [22], similar to Cyclon [35], NewsCast [21], or Gozar [30]. This service is periodically executed by all the nodes and provides each node with a random sample of the existing nodes in the overlay. This layer also enables nodes to propagate control information that are required by the upper layer. In particular, every node piggybacks its subscription information on the gossip messages that it sends out.

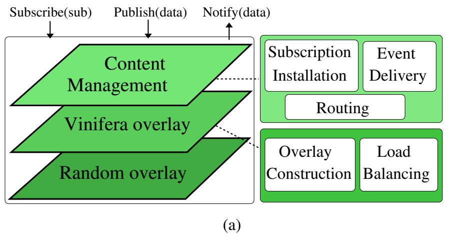
Vinifera overlay. The topology of this layer is constructed by capturing the existing subscription correlation in the system and clustering similar nodes together, using the same gossiping protocol. Moreover, we make this topology navigable by embedding it into an identifier space and enriching it by Small-World links following Kleinberg's model [24]. The resulting topology allows efficient routing while preserving interest locality.

Vinifera Content management layer. This layer consists of several components that work together to manage the efficient delivery of the content. It exploits the navigability and the interest locality of the underlying layer by constructing a forest of dissemination structures based on RV routing. Because of the inherent interest locality property of the underlying overlay, each and every dissemination tree is expected to be small and efficient, involving mostly the interested nodes in the dissemination process. At the same time, Vinifera trees are expected to be shorter as compared to the state-of-the-art.

IV. Vinifera

A. *Preliminaries*

As we mentioned in the previous section, Vinifera is a gossip-based protocol, i.e., each nodes periodically exchanges some information with some other nodes in the overlay. This



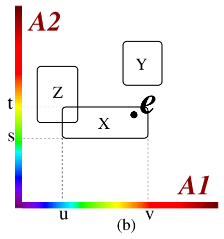


Fig. 1. (a) Vinifera architectural model (b) Vinifera Subscription model

information includes the node *profile*, which contains the *node identifier* and the node *subscription*. The node identifier is selected uniformly at random from a globally known identifier space. The subscription scheme includes n attributes from A_1 to A_n, of any type, where attribute A_i could take values between $v_{i_{min}}$ and $v_{i_{max}}$. We map the range $[v_{i_{min}}, v_{i_{max}}]$ to the entire node identifier space, by applying an order preserving hash function (OPHF) [13] over the values that are valid for each attribute. An OPHF guarantees that *if* $v > u$ *then* *OPHF(v)* $>$ *OPHF(u)*. For the sake of simplicity, from now on we refer to the hashed value *OPHF*(v), only as v. Each node subscribes in the system by introducing a number of constraints over a subset of attributes. A constraint specifies either an exact value (equality) or a range of values for an attribute, and a subscription S is the conjunction of all such constraints. Figure 1(b) shows a system with two attributes A_1 and A_2 and three subscriptions: X, Y, and Z. Subscriptions are shown by rectangles that specify the ranges over the two attributes. For example, subscription X is modeled as:

$$S_x : A_1 \in [u, v] \wedge A_2 \in [s, t]$$

A data item, or *event*, is a point in the attribute space, with exact values for all the attributes. An event *matches* a subscription, if each and every attribute value satisfies the corresponding constraint over that attribute. For example event e in Figure 1(b) matches subscription X, but it does not match subscriptions Y and Z.

B. Components

1) Overlay Construction: To enable nodes to select their *neighbors* (i.e., links or connections), based on their interest, identifier, or load, we employ a topology management protocol, inspired by T-Man [20], on top of the peer sampling service provided by the underlying random overlay. Each node p, maintains a *routing table*, i.e., a list of its neighbors, which it periodically exchanges with a neighbor, q, chosen uniformly at random among the existing neighbors in the routing table. Node p, then, merges its current routing table with the routing table of q, together with a fresh list of the nodes, provided by the underlying peer sampling service. The resulting list becomes the candidate neighbors list for p. Next, p selects a number of neighbors among the candidate neighbors and refreshes its current routing table. The same process will take place at node q.

Every vinifera node selects three types of links. First, each node maintains two links to connect to the nodes that are closest to it in the node identifier space, one in each direction.

Algorithm 1 Select Primary Attribute

```
1:  procedure SELECTPRIMARYATTRIBUTE
2:      for all $A_i$ in self.S do                    ▷ S represents node subscription
3:          rank($A_i$) $\leftarrow 0$                       ▷ initialize the rankings
4:      end for
5:      for all n in self.neighbors do
6:          for all $A_i$ in self.S do
7:              if n.S.contains($A_i$) then
8:                  self$C_i$ $\leftarrow$ self.S.getC($A_i$)          ▷ self constraint over $A_i$
9:                  n$C_i$ $\leftarrow$ n.S.getC($A_i$)             ▷ neighbor constraint over $A_i$
10:                 if overlapping(self$C_i$, n$C_i$) $\neq \oslash$ then
11:                     rank($A_i$) =rank($A_i$) + 1
12:                 end if                     ▷ increment the rank of the attribute
13:             end if
14:         end for
15:     end for
16:     $A_p \leftarrow A_i$ where rank($A_i$) is highest for all $A_i \in$ self.S
17: end procedure
```

These links are called *ring links*, because eventually these links shape up a ring structure in the overlay. The ring topology guarantees the existence of a path between any two nodes, and ensures the lookup consistency in the overlay, which is later required.

Next, to boost the routing efficiency, each node also selects some *small-world links*, based on the idea introduced by Kleinberg [24]. More precisely, node p selects node q as a small-world link, with a probability inversely proportional to the distance from p to q in the identifier space. It is proved that, having K_{sw} such neighbors enables a poly-logarithmic routing cost in the overlay $(O(\frac{1}{K_{sw}} log^2 N))$ [26].

Finally, links that are selected based on similarity of subscriptions are referred to as *friend links*. Every Vinifera node selects K_f friend links. These links connect together nodes with similar subscriptions. In a system with one attribute the similarity between two nodes p and q is captured by

$$Utility(p, q) = \frac{S_p \cap S_q}{S_p \cup S_q} \qquad \text{(Function I)} \qquad (1)$$

where, S_i contains the range(s) that node i has subscribed to. However, when there are more attributes, this approach is not readily applied. Instead, each node first selects one of the attributes, and then uses the mentioned utility function along that attribute only. As we will explain in Section IV-B4, when an event is published in a system with multiple attributes, multiple copies of the event are propagated in the overlay, one for each attribute. Therefore, to guarantee the event delivery, it is enough if a node is efficiently located in a cluster associated with only one of the attributes. The clusterization, i.e., the friend links selection, is completed in two steps:

- A node first examines the subscriptions of its candidate neighbors to select an attribute, across which it has more neighbors with overlapping ranges. We refer to this attribute as the *primary attribute* for the node. Algorithm 1 illustrates how the primary attribute is selected. The basic idea is that a node assigns a rank to each of the attributes in its own subscription. The rank of an attribute is calculated by counting the number of neighbors with an overlapping interest on that attribute (Lines 10, 11).

Algorithm 2 Range Query

```
 1: procedure LOOKUP(requester, lookupRequest)
 2:     if requester $\neq$ self then
 3:         installNeighborSubscription(requester, lookupRequest)
 4:     end if
 5:     RVNodes $\leftarrow$ NULL
 6:     if overlap(lookupRequest.range, [self, successor]) $\neq \oslash$ then
 7:         RVNodes.add(successor)
 8:     end if
 9:     for all neighbor in self.neighbors do
10:         if lookupRequest.range.includes(neighbor) then
11:             RVNodes.add(neighbor)
12:         end if
13:     end for
14:     if RVNodes $\neq$ NULL then                                  ▷ Shower
15:         for all RV in RVNodes do
16:             send lookup(self, lookupRequest) to RV
17:         end for
18:     else                          ▷ Proceed with a lookup for the beginning of the range
19:         nextHop $\leftarrow$ findNextHop(lookupRequest.range.from)
20:         send lookup(self, lookupRequest) to nextHop
21:     end if
22: end procedure
```

Finally, the attribute with the highest rank is selected as the primary attribute (Line 16).

- Next, the node uses the utility function (Function I) on the primary attribute and biases its neighbor selection towards selecting those nodes with higher rank as its friend links.

The combination of ring, small-world, and friend links results in a hybrid overlay, on top of which we build the data dissemination paths. We show, in the experiment section, that such a hybrid topology performs significantly better than a pure small-world overlay, as it not only reduces the unnecessary traffic in the network, but also improves the routing efficiency.

2) Routing: The basic routing or lookup in Vinifera is a greedy distance minimizing algorithm, i.e., at each step the lookup request is routed to a node which is closer to the destination. The destination of a lookup for value v is a node with the closest, but higher, identifier to v, which we refer to as the *rendezvous node (RV)* for v.

In Vinifera, nodes can not only route towards exact values, e.g., v, but also towards ranges of values, e.g., $[u, v]$ (Algorithm 2). To perform a range query, a node first applies an OPHF on the range boundary values. Then, a showering routing protocol [15] is executed, i.e., every node forwards the lookup request to as many nodes as it knows that fall into the range of the hashed values. Lines 15 to 17 in Algorithm 2 describe how the showering mechanism is performed. In case the node does not know any node in the requested range (line 18), it performs a simple greedy routing, i.e., it forwards the request to a node with closer, but not higher, identifier to the beginning of the range. The lookup ends at one or more consecutive RV nodes, each responsible for a part of the range.

3) Subscription Installation: Every node installs its subscription along the routing path to the rendezvous node(s) for the range over its primary attribute. We refer to this path (from the node itself to the rendezvous node(s)) as the *installation path*. Note that, we take advantage of the Vinifera overlay topology, by installing node subscriptions along the attribute for which they clustered more effectively, i.e., their

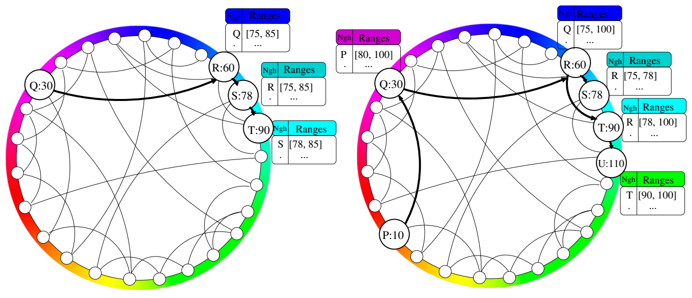


Fig. 2. (a) Subscription [75, 85] for node Q is installed. (b) Subscription [80, 100] for node P is installed (and aggregated with the previously installed subscription).

primary attribute. As a result, nodes that have the same primary attribute, and a similar range of interest over that attribute, will lookup for the similar rendezvous nodes. Since these nodes are very likely to be directly connected through friend links, they share most of their routing path towards the rendezvous node(s), with high probability. This is important, as it reduces the amount of traffic overhead transferred on the delivery paths.

Figure 2(a) illustrates the lookup process with a single attribute. Assume node Q wants to subscribe for the hashed values from 75 to 85. Among its neighbors, it selects node R, which has the closest, but not higher, node identifier to the requested range (Line 19 in Algorithm 2). The request is, therefore, sent from Q to R. Node R forwards the request to its neighbor S, which falls into the requested range (Line 10 in Algorithm 2). Node S takes the responsibility for the sub-range [75, 78], and also forwards a request for the remaining sub-range to node T (Line 6 in Algorithm 2). Consequently, the installation path from the subscriber node to the RV nodes is constructed (Path $Q \rightarrow R \rightarrow S \rightarrow T$ in Figure 2(a)).

Every node on the installation path maintains a table, called a *Content Routing Table* or *CRT* for short. The CRTs are populated when the queries are forwarded in the overlay. In our example, node R adds an entry to its CRT, for node Q requesting range [75-85], while node S registers range [75, 85] for node R. Finally node T registers a request from node S for the range [78-85]. In this example, we only have one attribute, and therefore CRTs, only include the requested range for that single attribute. If there are more attributes, each entry associates the requested range(s) with an attribute. Moreover, each entry of CRT contains the complete subscription(s) of the requesting node over all the attributes. This field will be used during event delivery process, described in Section IV-B4. Putting all these together, an entry of a CRT is a tuple in form of $< Ngh, Attr, Ranges, Subscriptions >$, where *Ngh* indicates the requesting neighbor, *Attr* is the requested attribute, *Ranges* are the interested ranges over the requested attribute, and *Subscriptions* are the subscription requests, containing all the attributes, which are received through the requesting neighbor.

The subscription installation process is further equipped with an aggregation technique. That is, whenever possible, a node aggregates all the requests it receives from the same neighbor on the same attribute. This is usually referred to as subscription subsumption or covering in the literature. For example, in Figure 2(b) node P appears in the system and subscribes for the range [80-100]. The closest neighbor of P to the requested range is node Q. So P sends a request to Q. As a result, Q installs this request in its CRT, and forwards it further to node R. Since node R previously had an entry for Q in its CRT (for the range [75-85]), it aggregates the two requests and modifies the entry for Q to range [75-100]. Now R knows two nodes, S and T, in the requested range. So it showers the request to both of them, by sending a request for range [75-78] to S and the remaining part of the range to T. When this new request is further forwarded in the overlay, nodes S and T similarly update their CRTs with the range [75, 78] and the aggregated range [78-100], respectively. Then, node T forwards the request further to node U, which is also a rendezvous node for a part of the requested range.

As mentioned previously, thanks to the employed clustering technique, nodes with similar subscriptions on the same primary attribute are grouped together. When these nodes install their subscriptions, they initiate a routing towards the same or close-by rendezvous nodes. Therefore, the installation tree is mostly shared between such nodes, thus, the requests can be effectively aggregated. This results in smaller CRTs, as well as less traffic overhead in the overlay. Smaller CRTs not only reduce the maintenance cost of the trees, but also simplify the matching process.

Note that, the subscription installation is a periodic process, and therefore, if a node fails or changes its subscription, it does not send any more request for the previously requested range, and therefore, the already installed rows in CRTs further ahead, can de-aggregate or be removed completely. This ensures that the quality of CRTs are constantly maintained.

4) Event Delivery: Any node in Vinifera can publish events. As mentioned previously, an event is a piece of data that has an exact value for each attribute. Therefore, in a system with n attributes, an event is associated with n rendezvous nodes, one for each attribute. When a node publishes an event $e\{v_1, v_2, .., v_n\}$, it sends one copy of the event to each of the n relevant rendezvous nodes, i.e., $RV(v_1)$, $RV(v_2)$, .., $RV(v_n)$, which are responsible for the values assigned to each of the attributes. This is done by initiating a simple rendezvous routing for each attribute. Then, n *delivery trees* for the event are constructed on the fly, by following the links on the reverse installation paths from the rendezvous nodes to the subscriber nodes, using the node CRTs. Note that each matching subscriber is registered in only one of the delivery trees, i.e., the one that corresponds to its primary attribute. So, it does not receive redundant messages from multiple trees.

The delivery trees are constructed as follows. Each rendezvous node, matches the event against the subscriptions that are registered in its CRT, and sends the event only to the neighbors with matching subscriptions. Note that, the matching is performed on the whole registered subscriptions, that is, if the event does not match the registered subscription of a node on any of the attributes (primary or not), the event will not be forwarded further to that node. Likewise, every node on the path performs such a matching process and forwards the event further if it matches the registered request, until it reaches the interested subscribers. By this approach, the matching process will be carried out as the event goes down to the subscribers, while every node maintains partial information about the other nodes. In essence, we distribute the load of matching events against subscriptions between the nodes that are on the installation path. At every step, those branches that are not interested in the event are pruned and the event is forwarded only down the paths that hold some interested node(s).

5) Load Balancing: Due to the prevalent skewed subscription patterns in the real world, the use of an OPHF inevitably results in a non-uniform identifier distribution and thus, an unbalanced load on the nodes. More precisely, some regions in the identifier space might be very popular with huge number of corresponding events, while some other regions might not be popular at all. So, the nodes who happen to fall into the popular regions may have to deal with huge number of requests. For example, if an attribute in the subscription scheme is temperature in a city, then the published events are most likely in the range $[-10, +30]$, probably a few around this range, and almost no event in less than -30 or over $+50$. Hence, node that have an identifier between OPHF(-10) and OPHF$(+30)$ are likely to be highly loaded, while the rest of the nodes are under loaded.

To alleviate this problem, we utilize an idea, inspired by [23], for adapting the node identifiers to the existing load in the network. The idea is that Vinifera nodes may change their identifiers along the ring, while their connections remains intact. In other word, nodes change their identifier to an identifier between themselves and their successor. The new identifier is assigned to the node to halve the load on the successor. Since nodes do not change their neighbors upon change of the identifier, they can easily inform their neighbors of the new identifier, when they send the next gossip message to the neighbors.

For our system, we define a measure of load as follows. Every node counts the number of events that it receives as a rendezvous node, without having any interest in them. Whenever the node sends its gossip message to its predecessor, it piggybacks its current load on the message. In order to prevent perturbation of the node identifiers, we define a threshold β for load imbalance between a node and its successor. When the difference of load between the two nodes exceeds the threshold β, the proceeding node changes its identifier to a value closer to its successor. Then the two nodes update their load to the average of their current loads. We show in the evaluation section that by employing this mechanism, Vinifera nodes can adapt to even very skewed user subscriptions.

C. Maintenance

In general, P2P networks are subject to churn, i.e., nodes join or leave the system continuously and concurrently, and network capacity changes. Therefore, it is essential to design P2P systems that tolerate failures. When a node fails in Vinifera, all the layers take the required actions to deal with that failure. The random overlay at the bottom, which is a gossip-based peer sampling service, is inherently failure tolerant. More details on fault tolerance in such protocols can be found in [22].

In the Vinifera overlay layer, failure handling is done similarly to the random overlay. As we described in Section IV-A, nodes periodically send their profile to their neighbors. This profile message, therefore, serves as a heartbeat message and enables the nodes to detect the failure of their direct neighbors. When a node fails, its direct neighbors that detect the failure, remove the entries for the failed neighbor from their partial view. When these nodes exchange their views with other nodes in the system, the contacted nodes will also receive the updated information and remove the stale entries, accordingly. Therefore, the information about the failed nodes, is propagated in the overlay by taking advantage of the ongoing gossiping protocol.

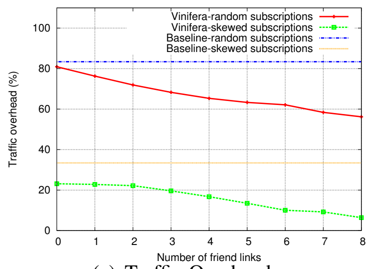
In the content management overlay, we need to ensure that CRTs are always updated and do not contain stale entries, i.e., when a node fails, we should remove its subscription from all the CRTs along the installation path. Note that, every entry in the CRT has an expiry timestamp. If a node does not receive a new subscription request from its neighbor, it will automatically remove the request from its CRT. Normally, in each gossip round requests are resent and therefore maintained in the CRTs. When a node fails, however, the first node met on the installation path, which has been directly connected to the failed node, detects the failure and removes the subscription of the failed node from its CRT. Therefore, it never again forwards the requests for that subscription, i.e., the next node on the relay path, will not receive the request for that subscription, thus, removes the entry from its CRT. Note that, if an entry in the CRT is the result of an aggregation, i.e., a node A received two requests from the same neighbor B for some overlapping ranges, and part of this aggregated range concerned a node C that is failed now, only the part that corresponded to C will be de-aggregated, as node B will send only one of the requests to node A, thereafter.

V. Evaluation

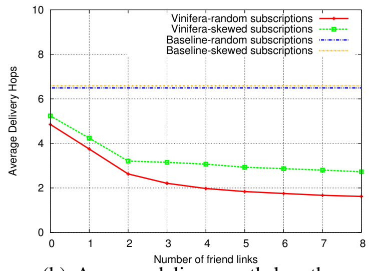
We implemented Vinifera in Peersim [29], a discrete event simulator for P2P applications. Through extensive simulations with a hybrid of cycle based and event based models, we evaluated the performance of Vinifera, while comparing it against a baseline system, inspired by the state-of-the-art techniques such as Ferry [39], eFerry [37], and CAPS [31]. That is, the baseline system is a pure small-world overlay, thus, requires a bounded node degree and guarantees a bounded routing time, but it is oblivious to node subscription, with no load balancing mechanism. In the lack of real-world traces, we synthetically generated user subscriptions, using a Zipf-like distribution over the attribute space [9]. Unless otherwise mentioned, the network size is 1000.

A. Topology Configuration

In this experiment, we investigate the choice of values for parameters K_{sw}, and K_f, which define the number of small-world links and friend links, respectively. The total number of links per node is set to 10, among which two are dedicated to ring links and the rest are used for small-world and friend links, i.e., $K_{sw} + K_f = 8$. Figures 3(a) and 3(b) show the traffic overhead and average delivery path length of Vinifera and the baseline system for two subscription models. Since the baseline system does not have any friend links, its topology does not change across the X-axis. Zero friend link in Vinifera generates a pure small-world topology, which is oblivious to node subscriptions, just like the baseline system. Hence, at this point both systems have the same topology and that is why with random subscriptions, the improvement in the traffic overhead in Vinifera is negligible. However at the same point, the average path length in Vinifera is decreased. This is due to the existence of many short delivery trees in Vinifera, as opposed to a single long delivery tree in Ferry. With skewed subscriptions, we can also improve the traffic overhead from over 32% to 22%. By adding more friend links, we take advantage of the subscription similarities and significantly improve both metrics at the same time. With 8 friends and skewed subscriptions, for example, the traffic overhead drops from 32% to less than 10%, while the average path length is decreased from around 7 to 3. This proves the huge potential of exploiting user subscription correlations, common in real-world scenarios. In the rest of our experiments, we set K_{sw} to 0 and K_f to $log(N) - 2$, where N is the number of nodes in the network. However, for the applications that require an upper bound guarantee on the number of delivery hops, we can add more small-world links.



(a) Traffic Overhead.



(b) Average delivery path length

Fig. 3. Performance with variable number of friend links

B. Scalability

To measure the scalability of Vinifera, we performed experiments with different number of nodes, as well as, different number of attributes. Figure 4(a) shows the traffic overhead of both systems. The performance of both systems is almost the same for different network sizes. However, the traffic overhead in the baseline system is more than 80% for random subscriptions, whereas it is reduced to 60% in Vinifera. Note that random subscriptions bring up the worst case scenario, for nodes can not effectively benefit from our clustering technique, due to the lack of correlation between user subscriptions. However, it is shown that a significant subscription correlation exists in real-world application [25][34]. When the user subscriptions are skewed, the traffic overhead in the baseline system drops to nearly half, while Vinifera reduces the traffic to almost one sixth of that of the random subscriptions. This shows that our data dissemination overlay is remarkably benefiting from the utilized clustering technique. Figure 4(b) shows the average delivery path length of both systems in terms of hop counts. Here again, the number of hops in Vinifera is nearly one third of that of the baseline system. However, the path length is slightly bigger with skewed subscriptions, because the overlay topology is clustered. With the random subscription model, however, the overlay better resembles a random network, thus, we observe a reduced path length.

Next, we observe the behavior of both systems when the subscription model includes more attributes. We have designed Vinifera to work with any dimensionality. Because of the lack of real-world traces we had to decide on the number of dimensions ourselves while carrying out the experiments. To be consistent (as well as being able to easily compare) with the existing state-of-the-art solutions we used most of the parameters (including dimensionality) from the related work papers. As we observed, most of the related work had reported results with 2, 3, or 5 attributes. We also show the results with up to 5 attributes. In higher dimensions, Vinifera still exhibits consistent results. Nevertheless, with our randomly generated events, the fraction of matching events drops sharply, thus, the measured values become insignificant, making the system hard to evaluate. This is due to a phenomenon, known as *the curse of dimensionality* in the literature [7]. However, since in real life the generated data is not uniformly spread in the data space and there exists a correlation between user subscriptions and the generated data [9][25][34], Vinifera is expected to function effectively in higher dimensions under realistic workloads.

Figure 4(c) shows when the subscription model is random, the traffic overhead of the baseline system remains at around 80%. This overhead starts from around 58% in Vinifera, but increases in higher dimensions, because in a random subscription scheme there exists very little similarities to be exploited, and therefore, the primary attribute is practically a random attribute for each node. As a result, the installation paths, and thus, the delivery trees are scattered in the overlay and nodes can not effectively cooperate in data dissemination. However, this overhead is still less than that of the baseline system. With the skewed subscription model, both systems behave significantly better. The overall improvement is again due to the skewed event publication in the system. However, the improvement in the baseline system with more attributes is because instead of having one single tree, more delivery trees are constructed, one for each attribute. Each node joins one of these trees, as a subscriber, and does not receive the events that are forwarded on other trees, unless it is a relay node in those trees. However, in Vinifera nodes with similar subscriptions are grouped together, and the delivery tree is shared between these nodes. Thus, the overall number of uninterested node on the delivery trees is reduced, thus, the traffic overhead drops to nearly one third of that of the baseline system.

The average delivery path length of the two systems are shown in Figure 4(d). The baseline system delivers the events slightly faster when there are more attributes. However, the average delivery path length in Vinifera is still by far better than the baseline system, even with 5 attributes in the subscription model, thanks to the utilized clustering technique. As soon as an event reaches a cluster of nodes with matching subscriptions, it is propagated inside that cluster very quickly.

We conclude that although both systems can accommodate any number of attributes in the subscription scheme, Vinifera exhibits a significantly better performance than the baseline system, specially in the presence of skewed subscriptions.

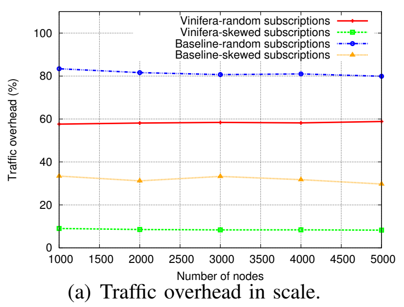
C. Load Balancing

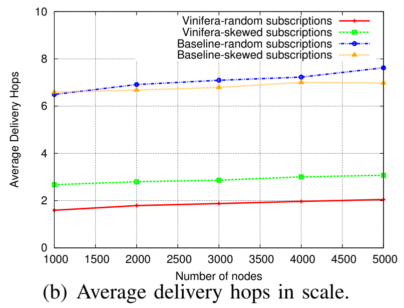
To explore how the load is distributed among the node in Vinifera versus the baseline system, we plot the cumulative distribution of load on the nodes, for 1, 2, and 3 attributes, and report the results in Figure 4(e). Although with more attributes, the load distribution is slightly degraded in Vinifera, it is still significantly better than the baseline system and the load on any Vinifera node never exceeds 30%. More precisely, over 95% of the nodes has a load less than 20%, even with 3 attributes, whereas, 10% of the baseline system nodes suffer from over 60% load in the system, while nearly 40% of the nodes have zero load. Figure 4(e) shows that the load in the baseline system is extremely unbalanced. This is because nodes up on the delivery tree are highly stressed, while leaf nodes are just receiving the service for free. There are nodes with even nearly 100% load (the rendezvous nodes), which can significantly harm the performance of the system as soon as they stop functioning correctly, due to congestion or failure. This problem prevents the baseline system to work under real-life scenarios where node and network failures are inevitable, while Vinifera can still function without having any imminent bottleneck.

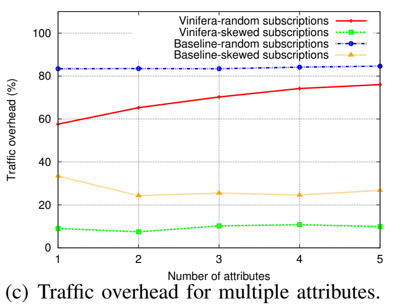
D. Workload

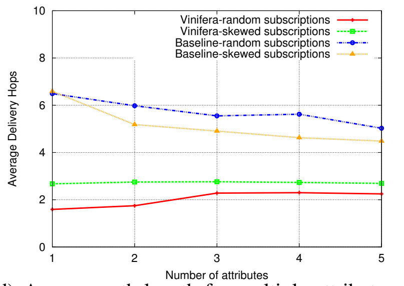
In this section, we examine if the systems can function under different workloads, i.e., under different event publication rates. To model the congestion, we assume that every node in the system can handle a bounded number of messages, X, in every round, and if it receives more events it will simply drop them. Then, we increase the number of events that are published in the system to up to five times X.

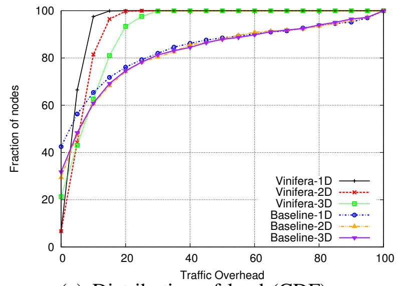
Figure 4(f) shows that the hit ratio in the baseline system significantly drops as soon as the publication rate passes the X threshold. Whereas, Vinifera survives even under high event publication rates. This is due to the fact that the baseline system relies on very few nodes to propagate the event in the system (only the intermediary nodes in the delivery tree). Therefore, under a high publication rate, those nodes become


(a) Traffic overhead in scale.


(b) Average delivery hops in scale.


(c) Traffic overhead for multiple attributes.


(d) Average path length for multiple attributes.


(e) Distribution of load (CDF).

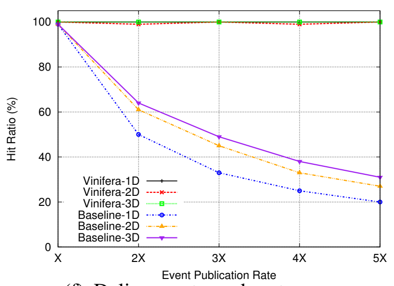
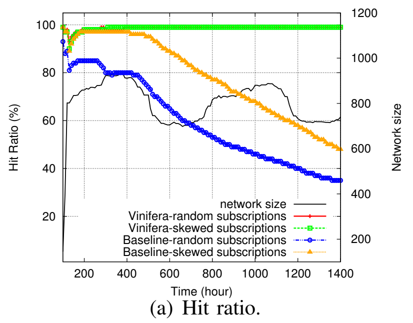
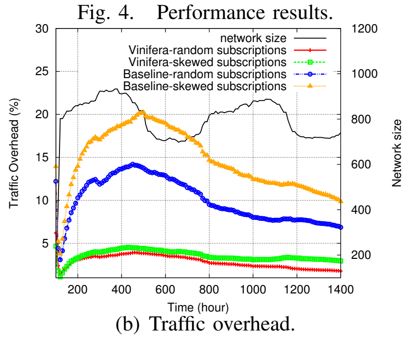

(f) Delivery rate under stress.

Fig. 4. Performance results.


(a) Hit ratio.


(b) Traffic overhead.

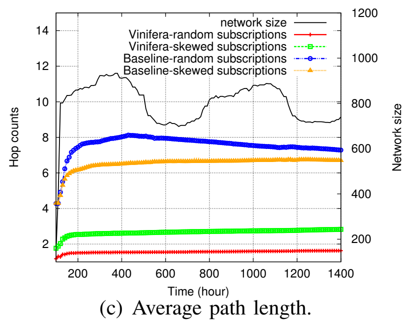

(c) Average path length.

Fig. 5. Performance under Skype churn trace.

highly overloaded and start dropping the messages. When a node in the tree drops a message, all its descendant nodes fail to receive the message. On the other hand, in Vinifera load is almost evenly distributed among the nodes. Thus, the nodes do not have to drop the messages due to the excessive load.

E. Fault Tolerance

To evaluate the performance of Vinifera in the presence of failures, we used real-world churn traces [18], that were obtained by monitoring a set of 4000 nodes participating in the Skype superpeer network for one month beginning September 12, 2005. In Figures 5(a) to 5(c) the x-axis shows the time, while the y-axis on the right shows the network size. The black solid line in the three graphs shows how the network size changes over time. Figure 5(a) shows the hit ratio of the two systems with random and skewed subscriptions. Although the hit ratio of Vinifera slightly decreases in the flash crowds, i.e., around time 100 h., when a large number of nodes join the system concurrently, the system recovers quickly and the hit ratio goes back to and remain at 100%, even in the presence of further joins and failures. In contrast, the hit ratio in the baseline system is highly affected by churn, due to the fragile structure of a single delivery tree. When this tree is broken, the baseline system can not repair it quickly enough to catch up with further event deliveries. When no more node joins or fails, the baseline system is potentially able to repair the dissemination tree. However, as we see in this real-world trace, this hardly happens.

Figure 5(b) shows the traffic overhead in both systems. The traffic overhead in Vinifera is one forth compared to the baseline system for both random and skewed subscriptions. Note that, the reduced traffic overhead in the baseline system is because it fails to deliver the events to all the nodes. We also observe, in Figure 5(c), that Vinifera is takes a four times shorter delivery path compared to the baseline system, in the presence of churn. Here again, we should take into account that in the baseline system some nodes are not receiving the events, and the measured values for the baseline system only include the nodes that received the events.

VI. Conclusion

We introduced Vinifera, a P2P content-based publish/subscribe system that enables users to subscribe for the information they are willing to receive, without having to rely on any single authority or central server. We employed a gossip-based technique to construct a topology that not only resembles a small-world network, but also connects the nodes with similar subscriptions together. On top of this hybrid overlay, we utilized a rendezvous routing mechanism to propagate node subscriptions in the overlay. Together with an order preserving hashing technique and an efficient showering algorithm we enabled range queries, and at the same time, we employed a load balancing technique to deal with the potential non-uniform user subscriptions. The combination of all these techniques are seamlessly integrated within a single gossiping layer, thus keeping Vinifera simple, lightweight and robust.

Our hybrid publish/subscribe system exhibited superior performance against the state-of-the-art techniques, effectively without the need to trade-off or degrade any important properties of the system. The overlay topology autonomously adapts to user subscriptions and is highly resilient to the dynamism in the network. The generated traffic overhead and the average delivery path length are simultaneously kept low, while only a bounded node degree is required and no global knowledge at any point is assumed.

References

[1] I. Aekaterinidis and P. Triantafillou. Pyracanthus: A scalable solution for dht-independent content-based publish/subscribe data networks. *Information Systems*, 36(3):655–674, 2011.

[2] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *Proc. of PODC'99*, pages 53–61. ACM, 1999.

[3] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proc. of P2P'02*, pages 33–40. IEEE, 2002.

[4] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *Proc. of DEBS'07*, pages 2–13. ACM, 2007.

[5] R. Baldoni and A. Virgillito. Distributed event routing in publish/subscribe communication systems: a survey, 2005.

[6] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proc. of ICDCS'99*, pages 262–272. IEEE, 1999.

[7] R. Bellman and R. Kalaba. On adaptive control processes. *IRE Trans. Auto. Cont.*, 4(2):1–9, 1959.

[8] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *Proc. of SIGCOMM'04*, pages 353–366. ACM, 2004.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proc. of INFOCOM'99*, volume 1, pages 126–134. IEEE, 1999.

[10] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[11] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *Proc. of DEBS'07*, pages 14–25. ACM, 2007.

[12] Y. Diao, S. Rizvi, and M. Franklin. Towards an internet-scale xml dissemination service. In *Proc. of VLDB'04*, pages 612–623, 2004.

[13] E. Fox, Q. Chen, A. Daoud, and L. Heath. Order-preserving minimal perfect hash functions and information retrieval. *ACM Trans. Info. Syst.*, 9(3):281–308, 1991.

[14] G. Fox and S. Pallickara. An event service to support grid computational environments. *Conc. and Comput.: Prac. and Exp.*, 14(13-15):1097–1127, 2002.

[15] Š. Girdzijauskas. *Designing peer-to-peer overlays: a small-world perspective*. PhD thesis, École Polytechnique Fédérale De Lausanne, 2009.

[16] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed. Magnet: practical subscription clustering for internet-scale publish/subscribe. In *Proc. of DEBS'10*, pages 172–183. ACM, 2010.

[17] E. Grummt. Fine-grained parallel xml filtering for content-based publish/subscribe systems. In *Proc. of DEBS'11*, pages 219–228. ACM, 2011.

[18] S. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer voip system. In *Proc. of IPTPS'06*, 2006.

[19] A. Gupta, O. Sahin, D. Agrawal, and A. Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Proc. of Middleware'04*, pages 254–273. Springer, 2004.

[20] M. Jelasity and O. Babaoglu. T-man: gossip-based overlay topology management. In *Proc. of the ESOA'05*, pages 1–15. Springer, 2006.

[21] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proc. of ICDCS'04*, pages 102–109. IEEE, 2004.

[22] M. Jelasity, S. Voulgaris, R. Guerraoui, A. Kermarrec, and M. Van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3), 2007.

[23] D. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proc. of SPAA'04*, pages 36–43. ACM, 2004.

[24] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proc. of STOC'00*, pages 163–170. ACM, 2000.

[25] H. Liu, V. Ramasubramanian, and E. Sirer. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *Proc. of IMC'05*, pages 3–3. USENIX, 2005.

[26] G. Manku, M. Bawa, and P. Raghavan. Symphony: distributed hashing in a small world. In *Proc. of USITS'03*, pages 10–10. USENIX, 2003.

[27] I. Miliaraki, Z. Kaoudi, and M. Koubarakis. Xml data dissemination using automata on top of structured overlay networks. In *Proc. of WWW'08*, pages 865–874. ACM, 2008.

[28] L. Ming, Y. Fan, K. Minkyong, H. C., and H. L. A scalable and elastic publish/subscribe service. In *Proc. of IPDPS'11*, pages 1254–1265. IEEE, 2011.

[29] A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *Proc. of P2P'09*, pages 99–100. IEEE, 2009.

[30] A. Payberah, J. Dowling, and S. Haridi. Gozar: Nat-friendly peer sampling with one-hop distributed nat traversal. In *Proc. of DAIS'11*, pages 1–14. Springer, 2011.

[31] J. Pujol-Ahullo, P. Garcia-Lopez, and A. Gomez-Skarmeta. Towards a lightweight content-based publish/subscribe services for peer-to-peer systems. *Int. Grid Util. Comput.*, 1(3):239–251, 2009.

[32] F. Rahimian, S. Girdzijauskas, A. Payberah, and S. Haridi. Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe. In *Proc. of IPDPS'11*, pages 746–757. IEEE, 2011.

[33] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of SIGCOMM'01*, pages 161–172. ACM, 2001.

[34] Y. Tock, N. N., H. A., and G. G. Hierarchical clustering of message flows in a multicast data dissemination system. In *Proc. of PDCS'05*, pages 320–326, 2005.

[35] S. Voulgaris, D. Gavidia, and M. Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Net. and Syst. Manag.*, 13(2):197–217, 2005.

[36] S. Voulgaris, E. Riviere, A. Kermarrec, and M. Van Steen. Sub-2-sub: Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks. In *Proc. of IPTPS'06*, 2006.

[37] X. Yang, Y. Zhu, and Y. Hu. Scalable content-based publish/subscribe services over structured peer-to-peer networks. In *Proc. of PDP'07*, pages 171–178. IEEE, 2007.

[38] R. Zhang and Y. Hu. Hyper: A hybrid approach to efficient content-based publish/subscribe. In *Proc. of ICDCS'05*, pages 427–436. IEEE, 2005.

[39] Y. Zhu and Y. Hu. Ferry: An architecture for content-based publish/subscribe services on p2p networks. In *Proc. of ICPP'05*, pages 427–434. IEEE, 2005.