Vitis: A Gossip-based Hybrid Overlay for Internet-scale Publish/Subscribe
Enabling Rendezvous Routing in Unstructured Overlay Networks

Fatemeh Rahimian[†‡], Sarunas Girdzijauskas[†], Amir H. Payberah[†‡], Seif Haridi[†‡]
[†]*Swedish Institute of Computer Science (SICS), Stockholm, Sweden*
[‡]*Royal Institute of Technology (KTH), Stockholm, Sweden*
Email: {fatemeh, sarunas, amir, seif}@sics.se

***Abstract*—Peer-to-peer overlay networks are attractive solutions for building Internet-scale publish/subscribe systems. However, scalability comes with a cost: a message published on a certain topic often needs to traverse a large number of uninterested (unsubscribed) nodes before reaching all its subscribers. This might sharply increase resource consumption for such relay nodes (in terms of bandwidth transmission cost, CPU, etc) and could ultimately lead to rapid deterioration of the system's performance once the relay nodes start dropping the messages or choose to permanently abandon the system. In this paper, we introduce *Vitis*, a gossip-based publish/subscribe system that significantly decreases the number of relay messages, and scales to an unbounded number of nodes and topics. This is achieved by the novel approach of enabling rendezvous routing on unstructured overlays. We construct a hybrid system by injecting structure into an otherwise unstructured network. The resulting structure resembles a navigable small-world network, which spans along clusters of nodes that have similar subscriptions. The properties of such an overlay make it an ideal platform for efficient data dissemination in large-scale systems. We perform extensive simulations and evaluate Vitis by comparing its performance against two base-line publish/subscribe systems: one that is oblivious to node subscriptions, and another that exploits the subscription similarities. Our measurements show that Vitis significantly outperforms the base-line solutions on various subscription and churn scenarios, from both synthetic models and real-world traces.**

I. Introduction

Publish/subscribe systems are nowadays widely used over the Internet. News syndication (RSS feeds), multi-player games, social networks such as Twitter or Facebook, media streaming applications, e.g., Spotify, or IPTV, are a few examples of such systems. Users of these systems express their interest in certain data, by subscribing to a number of topics, which can be daily news, a friend's tweets, a music playlist, or a channel on IPTV. Should any new data be published on a topic, the subscribers are notified and provided with the content. Depending on the application, this service could be bandwidth intensive and/or time critical, as in live streaming applications, or may include a large number of topics, as it is the case for Spotify playlists or social networks.

Currently, the majority of these systems use a client/server model and rely on dedicated machines to provide subscribe services. However, with a rapidly growing number of users on the Internet, and a highly increasing number of topics, it is becoming necessary to use decentralized models for providing such a service at a reasonable cost. Therefore, a lot of work has been done to design peer-to-peer publish/subscribe systems.

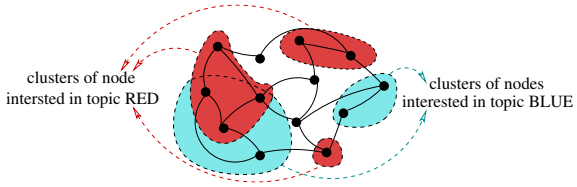


Figure 1. The biased neighbor selection puts together nodes with similar subscriptions. Due to bounded node degree, instead of a single cluster per topic, several disjoint clusters are formed. For example, red and blue topics have three and two clusters, respectively.

To provide the subscribe service, a range of solutions has been proposed. On the one extreme, nodes construct a separate overlay per topic, i.e., each node becomes a member of as many overlays as the number its subscriptions (e.g, Rappel [1] or Tera [2]). Although a node in these systems only receives the events that it has subscribed for, the number of the node's connections and, therefore, the overlay management cost grow linearly with the number of topics the node subscribes to. This, potentially, renders the system unscalable, when nodes subscribe to very large number of topics, e.g., thousands of topics, as is the case in some real world applications, e.g., [3].

At the other extreme, nodes use a bounded number of connections to manage all their subscriptions simultaneously (e.g, Scribe [4] or Bayeux [5]). These solutions, however, suffer from high *traffic overhead*, that is, nodes have to send and receive data in which they have no interest. As we will demonstrate later, in order to make sure all the subscribers of a topic receive their intended data, many nodes that are not subscribed for that topic have to get involved in data dissemination. Note that, although peer-to-peer users are generally willing to contribute their resources to the system, they might lose incentive to cooperate, if the amount of traffic they forward exceeds their expectations. For example,

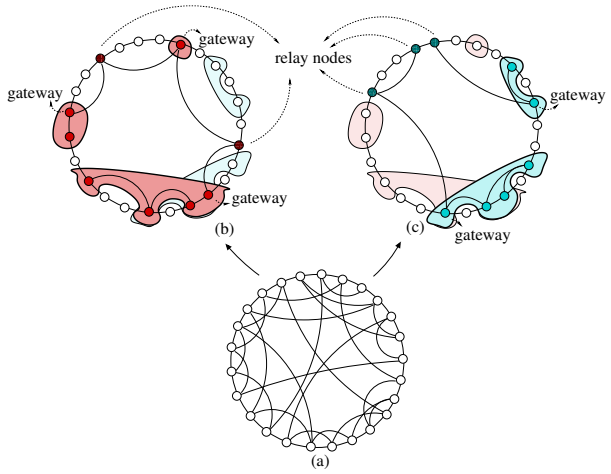


Figure 2. The resulting overlay is a single navigable small-world overlay (a), through which disjoint clusters of a topics connect together (b or c). The navigable small-world overlay enables relaying through the nodes, which are not subscribed to the topic, themselves.

a user of IPTV might permanently leave the overlay if it has to constantly forward a large media stream in which it has no interest. Therefore, it is crucial to decrease the traffic overhead of the nodes.

In this paper, we introduce Vitis, a topic-based publish/subscribe solution, that fills in the gap between the aforementioned extremes, while taking the best properties from both sides: bounded node degree and low traffic overhead. Vitis nodes run a gossip-based peer sampling service [6], to exploit the subscription similarities and select as neighbors, nodes with whom they share the most topics. Without the limitation on the node degree, a separate overlay per topic could eventually be formed. However, due to the bounded node degree, there is no guarantee that all the nodes, which are interested in a topic, connect together. In fact, any number of *clusters* for the same topic can emerge in different parts of the overlay (Figure 1). We denote a *cluster* for a topic as a maximal connected subgraph of the overlay, which includes a set of nodes that are all interested in that topic. Nodes inside a cluster are reachable from one another. In order to make sure a published event for a topic is delivered to all the subscribers, all the clusters of that topic must be linked together via other nodes. The path that connects different clusters of the same topic is called *relay path*. Such a path, includes nodes that are not interested in the topic themselves. We refer to these nodes as *relay nodes*, hereafter. The challenge is to decrease the number of required relay nodes, while making sure that all the clusters associated with a topic (and therefore, all the nodes interested in that topic) are linked together.

To enable relaying between the clusters, we introduce a novel technique for rendezvous routing [7] on top of an unstructured overlay. For that, Vitis nodes form a navigable small-world overlay (Figure 2), which is shown to have the best decentralized routing performance [8]. Then, nodes in each cluster select a number of representative nodes, as *gateways*. The number of gateways for a cluster is proportional to the diameter of the subgraph that represents the cluster. Gateway nodes are responsible for employing the navigable small-world overlay to connect to other clusters for the same topic. They perform a greedy lookup for the topic id, and all meet at the same node, i.e., rendezvous node. This approach is comparable to Scribe or Bayeux, but the difference is that nodes are efficiently grouped together in advance, and instead of each node independently performing the rendezvous routing, only few nodes, i.e., gateway nodes, establish the relay paths. In section III-B we elaborate on how the gateway nodes are selected and how the relay paths are established. We also show that the event propagation delay, in terms of the number of hops, is bounded to $O(\log^2 N)$, in our system. The resulting structure resembles a grapevine, with clusters of grape hanging from the canes, thus, inspired the name Vitis.

We evaluated the performance of Vitis through extensive large-scale simulations, with synthetic data as well as real-world subscription traces from Twitter [9], and churn traces from Skype [10]. We compare our system, with two base-line solutions: (i) a rendezvous routing system which is based on a structured overlay, with a bounded node degree, and oblivious to node subscriptions, and (ii) an unstructured solution that exploits the subscription correlation between nodes, without any bound on node degree. The results show that the traffic overhead in Vitis is between 40% to 75% less than the first base-line solution. We also show that, with a bounded node degree, Vitis always deliver the events to all the subscribers, while the hit ratio degrades in the second base-line solution, when the node degree can not grow indefinitely.

In the next section we describe the related work and position Vitis in the field. In section III we go into the details of the solution and in section IV we present the results of our experiments. We conclude the work in section V.

II. Related Work

The traditional architectures for publish/subscribe systems are the client-server and broker-based models. In systems based on either of these models, the subscriptions are submitted to a server (or broker). Also publishers send their events to this server (or broker), where the events are matched to the user subscriptions and forwarded to the users, accordingly. Solutions such as Siena [11], Gryphon [12], Hermes [13] or Corona [14] are in this category.

A more recent architecture for designing publish/subscribe systems, replaces the client-server or broker-based models with peer-to-peer overlays. This enables Internet-scale applications with many users as well as many topics. The peer-to-peer overlays can be roughly classified into two

main categories: structured and unstructured. Solutions such as Scribe [4] and Bayeux [5] are examples of structured overlay networks, while Tera [2], Rappel [1], StAN [15] and SpiderCast [16] fall into the second category, where a gossip-based approach is utilized. There are also solutions, like Quasar [17] or our solution, Vitis, which use gossiping to construct a hybrid of structured and unstructured overlays for event dissemination.

Regardless of how the overlay is constructed, the main challenge is to guarantee that nodes will receive all the events they have subscribed for, while not being overloaded with a large number of connections or excessive overhead. Tera [2], Rappel [1], StAN [15], and SpiderCast [16] construct a separate overlay for each topic. When a node subscribes to a topic, it becomes a member of that topic overlay. Therefore, published events for that topic are only distributed among the subscriber nodes and the traffic overhead is eliminated. However, nodes should join as many overlays as the number of topics they subscribe to. Thus, the node degree and overlay maintenance overhead grow linearly with the number of node subscriptions. This is, however, impractical for Internet-scale applications, when users subscribe to a large number of topics. We address this problem in Vitis, as nodes maintain a bounded number of connections, regardless of the number of their subscriptions.

To mitigate the scalability problem, SpiderCast [16] takes advantage of the similarity of interest between different nodes. The authors of SpiderCast argue that due to user subscription correlations, a single link can connect a node to more than one topic overlay. Thus, the number of required connections per node decreases. Since the user subscriptions are shown to be typically correlated in the real-world traces [18], [19], this idea works nicely with a limited number of node subscriptions. Nevertheless, the performance and scalability of SpiderCast is unknown, when the number of subscriptions is large or when there is churn in the environment. Moreover, any node in SpiderCast needs to have prior knowledge of at least 5% of other nodes in the system. In contrast, Vitis nodes do not need such a linear-scale amount of information about the other nodes in the system, and can subscribe to unbounded number of topics. In Section IV, we compare a SpiderCast-like system with Vitis and show that SpiderCast nodes suffer from maintaining a large number of connections, in order to receive all the events they have subscribed for.

There are also solutions that account for scalability by bounding the number of required connections per node, for example Quasar [17], which is a gossip-based solution, or Scribe [4] and Bayeux[5], which are DHT-based. In Quasar [17], each node exchanges with its nearby neighbors, an aggregated form of subscription information of itself and its neighbors a few hops away. Therefore, a gradient of group members for each topic emerges in the overlay. When a node publishes an event, targeted for a group, it sends multiple copies of the event in random directions along the overlay, and the event is probabilistically routed towards the group members. Quasar obviates the need for an overlay structure that encodes group membership information. However, it is inherently a probabilistic design model, even in a static environment. It also incurs high traffic overhead, since it is oblivious to nodes' subscriptions and involves many uninterested nodes in the event dissemination. In Vitis, on the other hand, we reach a full hit ratio, while minimizing the traffic overhead by organizing similar nodes into clusters.

In Scribe [4] or Bayeux[5], nodes are organized into a Distributed Hash Table (Pastry [11] and Tapestry [20], respectively), where each node maintains $O(\log N)$ connections. Then, a spanning tree is built for each topic, with a rendezvous node at the root, which delivers the events to the nodes that join the tree. This approach, however, forces many nodes to relay the events for which they have not subscribed, as they happen to be on the path towards the rendezvous node. Consequently, such systems suffer from a huge amount of traffic overhead. Vitis nodes also have a bounded node degree and form a tree-like structure per topic. However, unlike Scribe or Bayeux, the leaves in these trees are not single nodes, but groups of nodes, which are subscribed for that topic. We show through simulations, that an efficient clustering of nodes with similar interests, results in trees with far less intermediary nodes, and hence, much smaller traffic overhead.

Another solution, Magnet [21], exploits similar ideas of subscription correlation between the nodes, under the bounded node degree assumption. However, Magnet is purely based on a structured overlay and cannot fully capture the correlation between subscriptions, for it is bounded to one dimensional space, where the structured overlay is constructed. Also, Magnet is less robust in volatile environments, such as the Internet. In contrast, Vitis is not restricted to any dimension while capturing the subscription correlation (since clustering is done in an unstructured way) and as we show in our experiments, it is very robust due to the underlying gossip protocol.

Finally, there is recent work for resource location in clouds [22], which can be interpreted as a publish/subscribe system, though with quite clear differences. In [22], nodes query for a resource with certain attributes, and are redirected to a part of the cloud that contains the resources with requested properties. This work also employs a peer sampling service to build a structured and an unstructured overlay. In the unstructured overlay, resources with similar attributes are placed close to one another. However, [22] does not guarantee, and in fact does not need, that all the nodes with the queried properties are found. Nevertheless, in Vitis, we make sure that all the subscribers are found and informed of the published event. Moreover, [22] is not applicable for event dissemination, for it enforces a significant load on the nodes in the structured overlay.

III. VITIS

At a high level, Vitis borrows ideas from gossip based sampling services [6] (Section III-A) and rendezvous routing on structured overlays [7]. While benefiting from these ideas, Vitis employs a technique for selecting nodes that share topic interests (Section III-A2), and introduces a novel way of constructing a dissemination structure that minimizes the traffic overhead in the network (Section III-B).

Every Vitis node maintains a bounded-size *routing table (RT)*, which is a partial list of the existing nodes in the system that the node uses for routing the messages. The entries in the routing table are selected either as (i) small-world connections, or (ii) similarity connections based on a preference function. Hereafter, we refer to these two type of connections as *sw-neighbor* and *friends*, respectively. We also use the term *neighbor* to refer to any of the entries in the routing table, either friend or sw-neighbor.

Moreover, each node has a profile, which includes a unique node id, and the id of topics that the node subscribes to. Node ids and topic ids share the same identifier space and are generated by a globally known hash function that generates ids that are uniformly distributed in the identifier space, e.g, SHA-1. The topic id for topic t is denoted by $hash(t)$, hereafter. Subscribing to or unsubscribing from a topic, is done by adding or removing the topic id to/from the profile.

Every node periodically sends its profile to the nodes in its routing table, to inform them of its own subscriptions. This profile message also serves as a heartbeat message, and helps the nodes to constantly maintain their routing tables. When a node fails or leaves, its neighbors will stop receiving heartbeat messages and consequently, its entry will be removed from the routing table of its neighbors.

A. Neighbor Selection

Vitis utilizes a gossip-based peer sampling service to build a hybrid overlay. Any of the existing implementations for this service, e.g., [23], [6], [24], [25], can be used. When a node joins the overlay (Algorithm 1), it contacts a bootstrap node and receives a number of nodes to start communicating with. Then, the node runs the peer sampling service and periodically acquires fresh random samples of the existing nodes.

The overlay construction mechanism in Vitis is inspired by T-man [26], which is a generic protocol for topology construction and management. Each node, p, periodically exchanges its routing table (RT) 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 q's routing table, together with a fresh list of the nodes, provided by the underlying peer sampling service (Algorithms 2, lines 2-7). 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 (Algorithm 3). The core idea of our topology construction is captured in the neighbor selection mechanism, referred to as *selectNeighbors* in Algorithms 2 and 3 and described in Algorithm 4.

As mentioned previously, the routing table includes sw-neighbors and friend links. We define a system parameter k in Vitis, which determines the number of sw-neighbors in the routing table. The lower k is, the higher the upper bound on the routing cost is [27], while nodes are better grouped together and the traffic overhead decreases. That is, there is trade-off between the traffic overhead and the propagation

Algorithm 1 Join

```
1: procedure JOIN
2:     InitProfile()                                          ▷ subscribe to topics
3:     InitRoutingTable() ▷ get some neighbors from the bootstrap node
4:     start PeerSamplingService()
5:     do every δt                                            ▷ repeat periodically
6:         ExchangeRT()                                       ▷ Algorithm 2
7:         ExchangeProfile()                                  ▷ Algorithm 6
8: end procedure
```

Algorithm 2 T-Man - Active Thread

```
1: procedure EXCHANGERT
2:     neighbor ← selectRandomNeighbor()
3:     buffer ← getSampleNodes()      ▷ provided by the peer sampling service
4:     buffer.merge(RT)                                       ▷ RT is the local routing table
5:     Send [buffer] to neighbor
6:     Recv newBuffer from neighbor
7:     buffer.merge(newBuffer)
8:     RT ← selectNeighbors(buffer)
9: end procedure
```

Algorithm 3 T-Man - Passive Thread

```
1: procedure RESPONDTORTEXCHANGE
2:     Recv buffer from neighbor
3:     newBuffer ← getSampleNodes()
4:     newBuffer.merge(RT)
5:     Send [newBuffer] to neighbor
6:     newBuffer.merge(buffer)
7:     RT ← selectNeighbors(newBuffer)
8: end procedure
```

Algorithm 4 Select Neighbors

```
1: procedure SELECTNEIGHBORS(buffer)
2:     successor ← findSuccessor(buffer)
3:     buffer.remove(successor)
4:     selectedNeighbors.add(successor)
5:     predecessor ← findPredeccessor(buffer)
6:     buffer.remove(predecessor)
7:     selectedNeighbors.add(predecessor)
8:     sw-neighbor ← buffer.select-sw-neighbor(RANDOM-DISTANCE)
9:     buffer.remove(sw-neighbor)
10:    selectedNeighbors.add(sw-neighbor)
11:    for all node in buffer do
12:        utility[node] ← calculateUtility(node, self)
13:    end for
14:    sortedNeighbors ← utility[].sort()
15:    friends ← sortedNeighbors.top(RT-SIZE −3)
16:    selectedNeighbors.add(friends)
17:    return selectedNeighbors
18: end procedure
```

delay, which can be controlled by k. In Section IV we investigate the impacts of this trade-off on the performance of the system.

1) Sw-neighbor selection: In order to perform rendezvous routing [7], Vitis nodes establish sw-neighbors by utilizing a mechanism similar to Symphony [27]. Similar to Symphony, Vitis constructs a navigable small-world overlay, which guarantees a bounded routing cost that depends on the node degree. It introduces a distance function in the identifier space, where a neighbor for a node is selected with a probability that is inversely proportional to the distance between the two nodes.

The authors in [27] showed that selecting k links according to this probability function, results in a routing cost of the order $O(\frac{1}{k}\log^2 N)$ messages. For example, if one such neighbor is selected (as in Algorithm 4, line 8), the routing time is bounded to $O(\log^2 N)$. Note that, unlike Symphony, in Vitis nodes establish their sw-neighbors via periodic gossiping.

Moreover, our gossip protocol (Algorithms 2 and 3) enables Vitis nodes to form a ring topology in the identifier space. The ring is required for lookup consistency in the overlay, which is, in turn, required for constructing the relay paths (See Section III-B). Therefore, two entries of the routing tables are always dedicated for maintaining the neighbors on the ring. Each node selects two nodes with the closest id to its own, in the two directions, among the nodes it has learnt about so far, as its predecessor and successor on the ring (Algorithm 4, lines 2 and 6). Although initially the predecessors and successors may not be correctly assigned, T-Man protocol guarantees that through periodic gossiping the ring topology rapidly converges to a correct ring and is constantly maintained, thereafter [26].

2) Friend selection: The remaining candidate neighbors are ordered by a preference function. A node, then, selects the highest ranked nodes from this list (Algorithm 4, lines 11-15). The preference function takes into account: (i) the interest similarity of the nodes, as well as (ii) the event publication rate for different topics. It can also be extended to account for the underlying network topology and reduce the cost of data transfer in the physical network. The preference function, gives a pair-wise utility value to the nodes, according to the following function:

$$utility(i,j) = \frac{\sum_{t \in subs(i) \bigcap subs(j)} rate(t)}{\sum_{t \in subs(i) \bigcup subs(j)} rate(t)} \quad (1)$$

where $subs(i)$ indicates the set of topics that node i has subscribed to, and $rate(t)$ is the publication rate of topic t.

If the distribution of published events on different topics is uniform, nodes that have bigger interest overlap relative to the total number of their subscriptions, end up as friends. For example, if node p subscribes to topics $\{A, B, C\}$, node q subscribes to $\{C, D\}$, and node r subscribes to $\{C, D, E, F, G, H\}$, then $utility(p, q) = 0.25$, $utility(p, r) = 0.125$, and $utility(q, r) = 0.33$. That means, node p will prefer q to r, although it shares exactly one topic with both of them. Thus, node p less probably gets involved in the event propagation of events on topics $\{E, F, G, H\}$, in which it has no interest. Likewise, nodes q and r prefer to keep r and q in their local views, respectively.

If the publication rate varies for different topics, the interest overlaps are weighted by the publication rates. For example, if the publication rate for topic t goes to zero, i.e., almost no event is published on t, then t is practically ignored in the preference function. On the other hand, nodes will give a high utility to one another, if they are interested in a common topic that has a high rate of events.

B. Relay Path Construction

As we explained in Section III-A, the routing table size is bounded, thus, not all neighbors with utility greater than zero will be selected. As a result, instead of a unique cluster per topic, multiple disjoint clusters can emerge in the overlay. A cluster for topic t, is a maximally connected subgraph of the nodes that are all interested in t. If topic t has n disjoint clusters, these clusters are numbered and denoted as $C_i[t]$, where i is from 1 to n. To ensure that all n clusters of topic t are connected, some other nodes that are not subscribed to t have to get involved.

We define a *rendezvous node* for topic t, as a node with the closest id to $hash(t)$. Since Vitis constructs a small-world overlay, any node is able to route to any other node in the identifier space. To find the rendezvous node, a node performs a lookup on $hash(t)$, and all the nodes on the lookup path become *relay nodes* for t. This path, which we refer to as *relay path*, can include any kinds of links, e.g., friend, sw-neighbor or ring links (Figure 3). This is equivalent to the concept explored in Scribe [4] or Bayeux [5], where nodes subscribe on the path towards the rendezvous node and ultimately build a spanning tree.

In order to minimize the number of relay nodes for a topic, instead of letting each subscriber node route to the rendezvous node, as in, e.g., Scribe, nodes inside each cluster select a number of representative nodes, as *gateways*, to establish the relay path.

Algorithm 5 defines the gateway selection process. To select a gateway for cluster $C_i[t]$, each node in $C_i[t]$ initially proposes itself as gateway (Algorithm 5, line 3). This proposal is piggybacked on the node profile that is periodically sent to the neighbors (Algorithm 6). Likewise, the node receives other proposals from its neighbors, and revises its proposal for the next round (Algorithm 5, line 19). To avoid loops, each proposal also includes the node which proposed the gateway. This node is denoted as *parent* in Algorithm 5. Among the proposed gateways, the node

Algorithm 5 Update Profile

```
1:  procedure UPDATEPROFILE
2:      for all topic in profile.subscriptions do
3:          prop ← initProposal(self, self, 0)                ▷ (GW, parent, hops)
4:          for all neighbor in RT do
5:              if neighbor.isInterested(topic) then
6:                  new ← neighbor.getProposal(topic)
7:                  if neighbor = new.parent OR new.parent ∉ RT then
8:                      currentDis = distance(prop.GW, hash(topic))
9:                      newDis = distance(new.GW, hash(topic))
10:                     if newDis<currentDis AND new.hops+1 < d then
11:                         prop ← (new.GW,neighbor, new.hops+1)
12:                     end if
13:                     if new.GW=prop.GW AND new.hops+1 <prop.hops then
14:                         prop ← (new.GW, neighbor, new.hops+1)
15:                     end if
16:                 end if
17:             end if
18:         end for
19:         profile.subscriptions.update(topic, prop)
20:         if prop.GW = self then
21:             RequestRelay(topic)                           ▷ perform lookup(hash(t))
22:         end if
23:     end for
24: end procedure
```

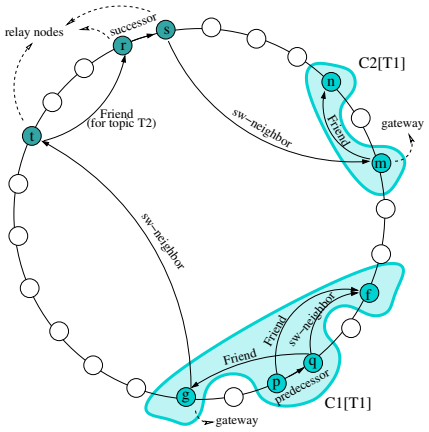


Figure 3. Node p publishes a notification inside its own cluster. The notification is flooded inside the cluster. It is also forwarded to the relay node t through the gateway g. The notification moves along the relay path up to the rendezvous node r, and then reaches the other existing clusters. Next, it is flooded inside those clusters.

selects as gateway the one that has the closest id to $hash(t)$, measured by *distance* function (Algorithm 5, lines 8 and 9). If the selected gateway, e.g., GW in Algorithm 5, is different from the current proposal, the node increases a counter inside the proposal for GW. This counter indicates the distance of the node to the GW, in terms of hop counts. If this distance exceeds a predefined threshold d, the node ignores the proposal (Algorithm 5, line 10). A gateway node, therefore, is responsible for the nodes, which are a maximum of d hops away from it. Consequently, the number of gateways per cluster becomes proportional to the diameter of the cluster, and can be controlled by the distance threshold d. That implies the worst case propagation delay inside a cluster is bounded to d. Hence, the propagation delay in Vitis is $O(\log^2 N + d)$. Nevertheless, d is a constant that does not depend on N and in all the practical scenarios, it can be set to a value less than $\log^2 N$. Therefore, the overall propagation delay is bounded to $O(\log^2 N)$. As our experiments show, in practice this value is much smaller than this upper bound.

When a node recognizes itself as gateway for topic t (Algorithm 5, line 20), it initiates the relay path construction by performing a lookup on $hash(t)$. Since all the lookups end up at the rendezvous node (the lookup consistency is ensured by the ring), all the clusters of topic t get connected.

It is important to note that nodes do not need to reach consensus on gateways and multiple gateways can be selected for each cluster. This results in establishment of several relay paths from the same cluster and, therefore, more traffic overhead. However, it does not affect the correctness of the solution and is beneficial because: (i) the overlay becomes more robust, in particular to the failure of gateway nodes or relay nodes along the path, and (ii) the propagation delay inside the cluster decreases, since the events are flooded simultaneously in different parts of the cluster.

Should a gateway node fail or disconnect from the cluster (e.g., due to a change of priorities that are enforced by the preference function), its immediate neighbors would detect the failure (after not receiving the heartbeat messages) and stop proposing it as a gateway. Therefore, in the proceeding rounds, those nodes select a different gateway.

C. Event Dissemination

Whenever a node publishes an event on a topic, it sends a notification to those neighbors in its routing table, which are interested in that topic, or act as a relay node for the topic. A node that receives a notification, pulls the event from the sender and forwards the notification to all its own interested neighbors. As a result, the notification propagates inside the cluster of the publisher node. When the notification is received by the gateway node, it is forwarded along the relay path. The notification goes up to the rendezvous node and again down the other existing relay paths, if any other cluster for that topic exists. It, then, reaches the gateway node(s) of those clusters, and will be flooded inside those clusters, accordingly. Figure 3 shows an example of how a notification is disseminated in the overlay. Node p publishes a new notification on topic t, and sends it to all its neighbors, which are interested in t. When this notification is received by the gateway node, g, it is forwarded on the relay path towards the rendezvous node, i.e., node t. When node t receives the notification, it sends it to the other existing relay path. Consequently, node m is informed and propagates the notification inside its own cluster. The event is pulled from the same path as the notification propagated along.

D. Overlay Maintenance

We use a mechanism similar to T-Man [26] and Scribe [4] for maintaining the routing tables and relay paths, respec-

Algorithm 6 Exchange Profile - Active

```
1:  procedure EXCHANGEPROFILE
2:      profile ← UpdateProfile()
3:      for all neighbor in RT do
4:          if neighbor.age > THRESHOLD then      ▷ remove the stale neighbors
5:              RT.remove(neighbor)
6:          else
7:              RT.neighbor.IncrementAge()
8:              Send [profile] to neighbor
9:          end if
10:     end for
11: end procedure
```

Algorithm 7 Exchange Profile - Reactive

```
1:  procedure RESPONDTOEXCHANGEPROFILE
2:      Recv profile from neighbor
3:      RT.update(neighbor, profile, 0)      ▷ 0 indicates the age of this neighbor
4:  end procedure
```

tively. Every time a node sends its profile to its neighbors, it increments the age of those neighbors (Algorithm 6). When it receives back a response from the neighbor, it marks that neighbor as fresh, by reseting its age to zero (Algorithm 7, line 3). After a predefined threshold, the stale entries are removed from the routing tables. This threshold determines the failure detection speed. The lower the threshold, the faster the failure detection is. However, if the threshold is too low, then the rate of false positives, due to the congestion in the network and varying link delays, increases. By increasing the threshold, the responsiveness of the failure detection can be traded off for more accuracy.

As we described earlier, the overlay is constructed by gossiping. Through gossiping, clusters are formed, gateway nodes are selected, and relay paths are established. The overlay maintenance is conducted in exactly the same way. When a node leaves the system or modifies its subscriptions, the friend selection mechanism in the proceeding rounds captures this change and routing tables are updated accordingly. If the node is a gateway, then its direct neighbors in the corresponding cluster will notice the change and revise their proposals for selecting a new gateway. If the node is a relay node or rendezvous node, the proceeding lookups by their neighbors on the relay path, will return a substitute node. Consequently, the overlay adapts to the changes in the network, while nodes constantly acquire fresh information through their neighbors.

IV. EXPERIMENTS

We implemented Vitis and two base-line solutions in Peersim [28], a simulator for modeling large scale peer-to-peer networks. The base-line solutions are:

- RVR: a structured RendezVous Routing solution that builds a multicast tree per topic, equivalent to that of Scribe [4] or Bayeux [5], with fixed node degree.
- OPT: an unstructured subscription aware solution that constructs an Overlay Per Topic, while minimizing node degrees by exploiting the subscription correlations, similar to SpiderCast [16].

To make the three systems comparable they use the same peer sampling service (Newscast [25]) and overlay construction protocol (T-Man [26]).

We evaluate Vitis against RVR and OPT with subscription patterns, generated from a synthetic model as well as real-world Twitter traces [9]. We investigate the impact of varying publication rates and routing table sizes on the performance of the systems. Moreover, the robustness of Vitis under churn is evaluated by utilizing traces from Skype [10].

In our simulations, we measure the following metrics:

- *Hit ratio*: The fraction of events, on all topics, that are received by the subscriber nodes;
- *Traffic overhead*: The proportion of relay (uninteresting) traffic that nodes experience;
- *Propagation delay*: The average number of hops events take to reach to all the subscriber nodes.

A. Experimental settings

We measure the performance of Vitis, RVR, and OPT with 10,000 nodes. Unless otherwise mentioned, k is set to 3, the routing table size is set to 15, d is set to 5, and different topics have the same rate of publication.

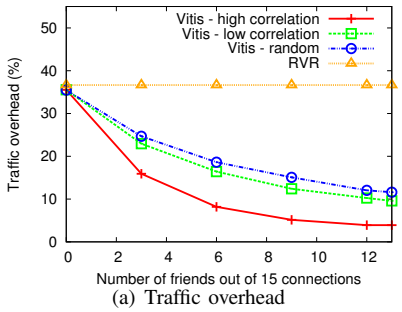
We generate three subscription patterns to model different levels of interest correlation. This data generation model was inspired by a work of Wong et al [29]. The subscription patterns are:

- *Random*: nodes select 50 out of 5000 topics uniformly at random;
- *Low correlation*: nodes group 5000 topics into 100 buckets and select 50 topics uniformly at random from 5 different buckets (10 topics from each bucket);
- *High correlation*: nodes group the 5000 topics into 100 buckets and select 50 topics uniformly at random from 2 buckets (25 topics from each bucket).

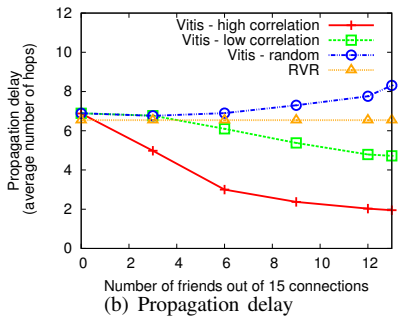
Note that, in all the above subscription patterns, the average topic popularity, i.e., the population of nodes subscribed to a topic, is uniform. Whereas, the distribution of interest correlation, captured by Equation 1, is different in the three patterns. Since RVR exhibits similar behavior with random and correlated subscriptions, we draw only a single line for it in the plots. Moreover, since SpiderCast is targeted for real-world scenarios with high subscription correlation, we investigate the performance of OPT only with Twitter subscriptions.

B. Friends Vs. sw-neighbors

In this experiment, we investigate the performance impact of varying the number of friends versus sw-neighbors. We bound the node degree to 15, that is, each node has a routing table of size 15, among which two links are dedicated for the predecessor and the successor of the node. That means,

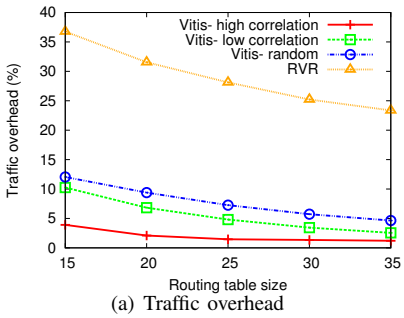


(a) Traffic overhead

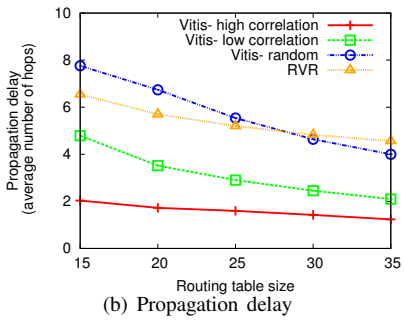


(b) Propagation delay

Figure 4. Measurements with varying number of friends



(a) Traffic overhead



(b) Propagation delay

Figure 6. Measurements with different routing table sizes

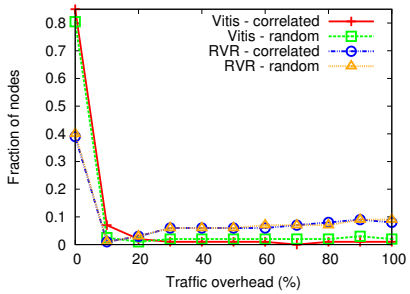


Figure 5. Distribution of traffic overhead

nodes have at least two sw-neighbors in all the experiments. The rest of the links can be selected, either as a friend or as sw-neighbor.

The results showed that both Vitis and RVR have 100% hit ratio in all settings. As we observe in Figure 4(a), when more friends are selected, the traffic overhead in Vitis drops significantly. With correlated subscriptions, this traffic reduced by a factor of 88%. Even when the subscriptions are random, the traffic overhead in Vitis is less than one third compared to that of RVR. That shows Vitis is able to exploit even the slightest similarities between nodes subscriptions.

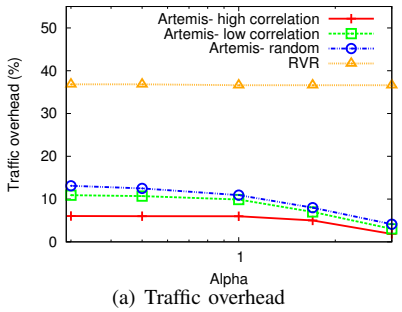
As it is shown in Figure 4(b), nodes with correlated subscriptions experience a better delivery time as well. The propagation speed improves when more friend links are selected. This is due to the fact that selecting more friends results in a better clustering of nodes with similar subscriptions. Thus, instead of having many small clusters, the overlay moves towards having fewer, but bigger clusters. Since the events are very quickly disseminated inside clusters (by flooding), most of delay is caused by the inter-cluster routing. Therefore with fewer clusters, the event dissemination happens much faster. For random subscriptions, however, the overlay ends up having multiple small clusters per topic. Therefore, inter-cluster routing plays an important role for delivering the events to the subscriber node. Since replacing sw-neighbors with friend links degrades the navigability of system, the improved traffic overhead in this case, comes at the cost of higher propagation delay. However, as discussed in section III-A, the propagation delay in our system is bounded to $O(\log^2 N)$.

Moreover, one might argue that although the average traffic overhead is reduced in Vitis, a high load is imposed upon gateway nodes, rendezvous nodes, or other relay nodes. Therefore, we show the traffic overhead distribution among the nodes in the overlay. Figure 5 shows that while the fraction of nodes with 10% overhead is increased, the fraction of nodes that have an overhead more than 20%, drops to less than one third in Vitis, compared to that of RVR. This shows that Vitis, not only reduces the average traffic overhead, but also improves the distribution of this traffic among the nodes.

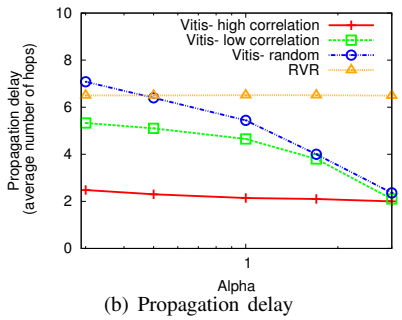
In the rest of our experiments we set one predecessor, one successor, and one sw-neighbor for each node. The rest of the links are selected as friends.

C. Changing the routing table size

In this experiment we compare the performance of Vitis to RVR, while changing the routing table size from 15 to



(a) Traffic overhead



(b) Propagation delay

Figure 7. Measurements with different publication rates

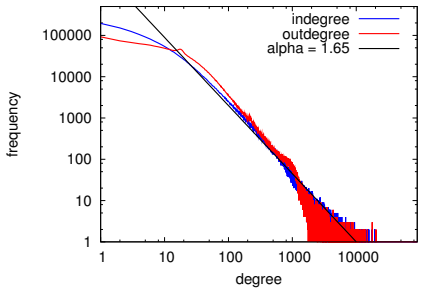


Figure 8. Distribution of in-degree and out-degree in Twitter

	In-degree (followers of a node)	Out-degree (subscriptions of a node)
Average	87.8797	87.8778
Variance	965262.3440	171085.2826
Std. Deviation	982.4776	413.6245
Max	349573	172268

Figure 9. Summary of statistical analysis of available Twitter data set

35. As it is shown in Figure 6(a), when nodes maintain bigger routing tables, the traffic overhead, as well as the propagation delay, decreases in both systems, though, for different reasons. In RVR, this improvement is because the rendezvous routing performs better, i.e., in fewer number of hops, with more small-world links. Thus, more efficient spanning trees with less intermediary nodes are constructed. In Vitis, however, the number of sw-neighbors are fixed and the additional entries in the routing tables are used for adding friend links. Therefore, nodes are grouped together more efficiently and fewer relay paths per topic are required. This means inter-cluster routing constitutes a smaller part of the event dissemination. This explains why event delivery latency in Vitis with random subscriptions, outperforms the RVR system, when the routing table size exceeds 30 entries.

D. Changing the publication rate

So far we have assumed a uniform distribution of published events on each topic. However, the publication rate of topics does not have to be uniform. In fact, usually there are a few hot topics with a high rate of publications, while other topics have a low publication rate. In this experiment, we show how our solution adapts to different publication rates. We employ a power-law function, with a parameter α, to define the distribution of events rate on different topics. We change α from 0.3 to 3, and evaluate the behavior of Vitis versus RVR. Note that the X-axis in Figure 7 is in the log scale. When α is close to 0.3, the distribution is similar to a uniform distribution as in the previous experiments. However, when α increases the distribution becomes more skewed. In the extreme case, when α is 3, almost all the events are published on a single topic.

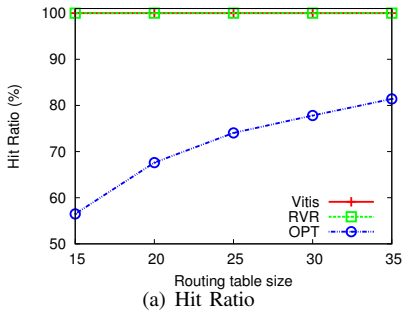
When the publication rate for different topics becomes more skewed, Equation 1 gives a higher utility value to the nodes that are interested in the hot topics. Thus, such topics end up having fewer and better connected clusters. This effect is similar to when the correlation level is increased. That is why in Figure 7, the performance of the scenario with random subscriptions gets closer to that of the scenario with high correlation, when α is increased.

Note that, while hot topics are prioritized, topics with less events might experience higher traffic overhead and propagation delay. However, since hot topics constitute most of the published events, and they are propagated efficiently, an overall improvement is achieved.

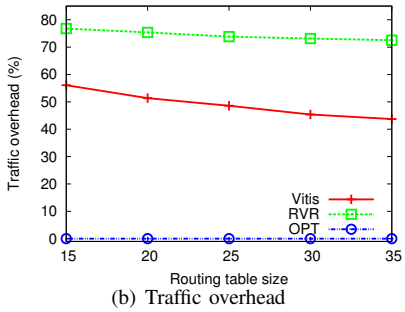
E. Real world subscriptions

In this experiment, we evaluate Vitis with both RVR and OPT. We use a subscription pattern extracted from nearly 2.4 million Twitter users [9]. Each node in Twitter plays a dual role, that is, it can follow (subscribe to) other nodes, and it can be followed by others (as a topic). Thus, both topics and nodes refer to the users of the system. We analyzed the available data set and came up with the statistical results reported in Figure 9. The distribution of nodes in-degree and out-degree are modeled by a power-law distribution with an estimated parameter of 1.65 (Figure 8).

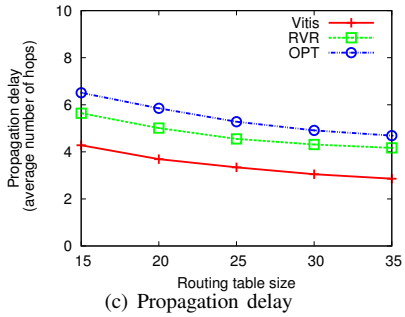
We took a sample of nearly 10000 nodes, by performing multiple breath first searches (BFS) [30]. Initially we randomly selected a number of nodes from the dataset. Then we added to this sample, all the subscriptions of these nodes, i.e., nodes being followed by the selected nodes. Next, we extracted all the relations (following or being followed) between these nodes. Finally, we removed subscriptions to the nodes outside the sample. In order to ensure that this



(a) Hit Ratio



(b) Traffic overhead



(c) Propagation delay

Figure 10. Measurements with Twitter subscription patterns

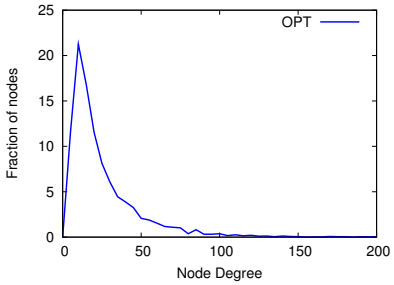


Figure 11. Node degree distribution in OPT

approach preserves the properties of the complete log, we took several samples and the similarity of in-degree and out-degree distribution of the samples and that of the full log was confirmed.

Unlike our previous configurations, in these experiments the number of subscriptions per node is not the same. We changed the routing table size from 15 to 35 and investigated the impact on the hit ratio, traffic overhead and propagation delay. Moreover, we measured the hit ratio for a node, 10 seconds after the node joins the system. That means a node is expected to receive the subscribed-to events, which are published 10 seconds after its joining time.

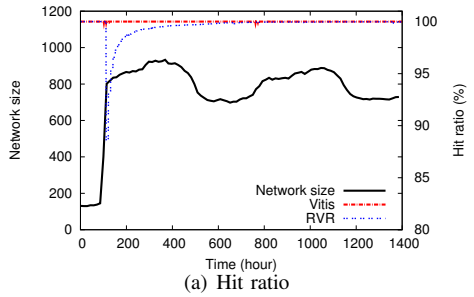
Figure 10(a) shows that the hit ratio in Vitis and RVR is 100%, while OPT with a bounded node degree can not achieve a full hit ratio. Even when the node degree is 35, OPT can only hit 80% of the subscribers, on average. In order to reach a 100% hit ratio, OPT needs to be free of any bound on the node degree. We performed another experiment to investigate the performance of OPT with unbounded node degree, and plotted the node degree distribution in Figure 11. As can be seen in this figure, more than two third of the nodes have a degree higher than 15. Also, 0.3% of nodes have a degree higher than 200 (Maximum observed degree is 708), which is not shown in the figure. This implies OPT-like solutions, that only rely on exploiting subscription correlations, can not scale in real world scenarios.

In contrast, OPT outperforms Vitis and RVR with respect to traffic overhead. Since OPT constructs a separate overlay per topic, the events are only disseminated among the subscribers and there is no traffic overhead at all. Figure 10(b) shows the traffic overhead of the three systems. As it is shown, Vitis and RVR has a higher level of overhead compared to Figure 6(a), which is due to the increased number of subscriptions (on average 80 subscriptions per node). Also, the number of topics in this experiments is doubled, since there are as many topics as the number of nodes. Therefore, the average population of nodes that are interested in a topic is less than the previous experiments. However, even with only 15 links per node, Vitis has 30% less traffic overhead compared to RVR. With 35 links per node, the traffic overhead in Vitis decreases to 43%, which is 40% better than RVR.

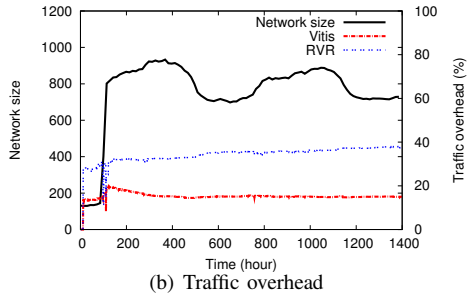
The propagation delay in all three systems exhibits a similar trend when the routing table size increases (Figure 10(c)), while Vitis is more than 1.5 times faster than RVR and 1.7 times faster than OPT. Note that due to the navigable structure, the delay in Vitis and RVR is bounded. However, a topic overlay in OPT might be any arbitrary graph and therefore there is no upper bound on the propagation delay.

F. Vitis under churn

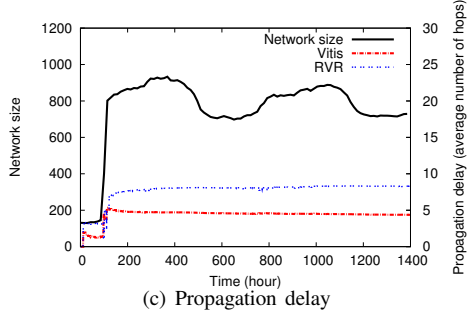
In this experiment, we use a scenario with churn, i.e., a scenario in which nodes can join or leave at any time. We use a real world trace [10], which monitors a set of 4000 nodes participating in the Skype superpeer network for one month beginning September 12, 2005. The routing table size is bounded to 15, and a uniform publication rate for the topics is considered. Like the previous experiment, the hit ratio for a node is calculated 10 seconds after the node joins the system. We compare Vitis with RVR and observe that,



(a) Hit ratio



(b) Traffic overhead



(c) Propagation delay

Figure 12. Measurements with Skype trace for churn in the network

due to the underlying gossip mechanism, both solutions react nicely to the churn and adapt to the changes of network.

As Figure 12(a) shows, although both systems can tolerate moderate churn, under *flash crowds*, i.e., a large number of nodes join at nearly the same time, the hit ratio in RVR goes down to 87%. That is because the stabilization time takes longer, and while the structure is not converged to a connected subgraph per topic, nodes may miss some events. This effect is also observable in our system. However, the worst case hit ratio is about 99%. This is because as soon as a node finds a group-mate for a topic, it can receive the corresponding events on that topic, without the need for establishing a relay path independently.

We also observe in Figure 12(b) that the traffic overhead in both systems does not change much over time. However, under flash crowds, the traffic overhead in RVR drops sharply. This is not an advantage though, because the relay paths are not established properly and nodes are missing their desired events (that is why the hit ratio drops as well). In contrast, the traffic overhead in Vitis slightly increases under flash crowds, because nodes inside the groups are not yet informed about their group-mates and therefore several gateway nodes start to build up the relay paths towards the rendezvous point. After a while, however, when the churn is moderate, the number of gateways and, consequently, the traffic overhead decrease. Likewise, Figure 12(c) shows that the propagation delay does not change in moderate churn. However, the increased level of delay after the flash crowd is due to the bigger size of the network.

V. Conclusion

We presented Vitis, a topic-based publish/subscribe system, which scales with the number of nodes as well as the number of topics in the overlay.

The main contribution of this paper is a novel hybrid publish/subscribe overlay that exploits two ostensibly opposite mechanisms: unstructured clustering of similar peers and structured rendezvous routing. We employ a gossiping technique to embed a navigable small-world network, which efficiently establishes connectivity among clusters of nodes that exhibit similar subscriptions. We also give a theoretical bound on the worst case delay.

We showed that Vitis fills in the gap in the range of solutions, by simultaneously achieving both bounded node degree and low traffic overhead. We evaluated Vitis, in simulations, by comparing its performance against two base-line solutions, which represent two main groups of the related work: a structured overlay that uses rendezvous routing (RVR), and a solution that takes advantage of subscription similarities to constructs an overlay per topic (OPT).

We used synthetic data as well as real-world traces from Twitter to model users subscriptions. We also used traces from Skype to show that Vitis is robust in the presence of churn. We showed that although exploiting subscription correlations results in great advantages, solutions such as OPT, which solely rely on such correlations, can not scale when the number of node subscriptions increases. Consequently, in real world scenarios, such solutions cannot guarantee that the subscribers receive their intended data, unless the node degrees are unbounded. In contrast, Vitis and RVR always reach a perfect hit ratio. This, however comes at the cost of some traffic overhead. We showed that, compared to the rendezvous routing solution, Vitis reduces the traffic overhead to less than 75% with synthetic data and 40% for real-world traces, while it speeds up the event dissemination in the overlay. Moreover, Vitis adapts to biased rates of events that are published on different topics, and builds more efficient groups for hot topics, thus, improving the overall performance of the event dissemination.

References

[1] J. Patel, É. Rivière, I. Gupta, and A. Kermarrec, "Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems," *Computer Networks*, 2009.

[2] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni, "TERA: topic-based event routing for peer-to-peer architectures," in *Proceedings of the international conference on Distributed event-based systems*. ACM, 2007.

[3] B. Krishnamurthy, P. Gill, and M. Arlitt, "A few chirps about twitter," in *Proceedings of the first workshop on Online social networks*. ACM, 2008.

[4] 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*, 2002.

[5] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. ACM, 2001.

[6] M. Jelasity, S. Voulgaris, R. Guerraoui, A. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems (TOCS)*, 2007.

[7] R. Baldoni and A. Virgillito, "Distributed event routing in publish/subscribe communication systems: a survey," *DIS, Universita di Roma La Sapienza, Tech. Rep*, 2005.

[8] J. Kleinberg, "The small-world phenomenon: an algorithm perspective," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. ACM, 2000.

[9] W. Galuba, K. Aberer, D. Chakraborty, Z. Despotovic, and W. Kellerer, "Outtweeting the Twitterers-Predicting Information Cascades in Microblogs," in *3rd Workshop on Online Social Networks (WOSN10)*, 2010.

[10] S. Guha, N. Daswani, and R. Jain, "An experimental study of the skype peer-to-peer voip system," in *Proceedings of IPTPS*. Citeseer, 2006.

[11] A. Carzaniga, D. Rosenblum, and A. Wolf, "Achieving scalability and expressiveness in an internet-scale event notification service," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, 2000.

[12] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward, "Gryphon: An information flow based approach to message brokering," in *International Symposium on Software Reliability Engineering*. Citeseer, 1998.

[13] P. Pietzuch and J. Bacon, "Hermes: A distributed event-based middleware architecture," *22nd International Conference on Distributed Computing Systems Workshops*, 2002.

[14] V. Ramasubramanian, R. Peterson, and E. Sirer, "Corona: A high performance publish-subscribe system for the world wide web," *Proceedings of Networked System Design and Implementation (NSDI)*, 2006.

[15] M. Matos, A. Nunes, R. Oliveira, and J. Pereira, "StAN: Exploiting Shared Interests without Disclosing Them in Gossip-based Publish/Subscribe," in *Proceedings of IPTPS*, 2010.

[16] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication," 2007.

[17] B. Wong and S. Guha, "Quasar: A Probabilistic Publish-Subscribe System for Social Networks," in *Proceedings of IPTPS*, 2008.

[18] H. Liu, V. Ramasubramanian, and E. Sirer, "Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews," in *Proc. of ACM Internet Measurement Conference*, 2005.

[19] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky, "Hierarchical clustering of message flows in a multicast data dissemination system," in *IASTED PDCS*, 2005.

[20] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," *Computer*, 2001.

[21] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed, "Magnet: practical subscription clustering for Internet-scale publish/subscribe," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*. ACM, 2010.

[22] J. Alveirinho, J. Paiva, J. Leitao, and L. Rodrigues, "Flexible and Efficient Resource Location in Large-Scale Systems," *The 4th ACM SIGOPS/SIGACT Workshop on Large Scale Distributed Systems and Middleware*, 2010.

[23] M. Jelasity, A. Kermarrec, and M. van Steen, "The peer sampling service: Experimental evaluation of unstructured gossip-based implementations," in *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. Springer-Verlag New York, Inc. New York, NY, USA, 2004.

[24] S. Voulgaris, D. Gavidia, and M. Van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, 2005.

[25] M. Jelasity and A. Montresor, "Epidemic-style proactive aggregation in large overlay networks," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*. IEEE Computer Society, 2004.

[26] M. Jelasity and O. Babaoglu, "T-Man: Gossip-based overlay topology management," *Lecture Notes in Computer Science*, 2006.

[27] G. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world," in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems-Volume 4*. USENIX Association, 2003.

[28] M. Jelasity, A. Montresor, G. Jesi, and S. Voulgaris, "PeerSim:A peer-to-peer simulator," *http://peersim.sourceforge.net*.

[29] T. Wong, R. Katz, and S. McCanne, "An evaluation of preference clustering in large-scale multicast applications," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.* IEEE, 2002.

[30] M. Kurant, A. Markopoulou, and P. Thiran, "On the bias of BFS," *Arxiv preprint arXiv:1004.1729*, 2010.