SpiderCast: A Scalable Interest-Aware Overlay for Topic-Based Pub/Sub Communication

Gregory Chockler Roie Melamed
Yoav Tock
IBM Haifa Research Laboratory
chockler,roiem,tock@il.ibm.com

Roman Vitenberg
Department of Informatics,
University of Oslo, Norway
romanvi@ifi.uio.no

ABSTRACT

We introduce SpiderCast, a distributed protocol for constructing scalable churn-resistant overlay topologies for supporting decentralized topic-based pub/sub communication. SpiderCast is designed to effectively tread the balance between average overlay degree and communication cost of event dissemination. It employs a novel coverage-optimizing heuristic in which the nodes utilize partial subscription views (provided by a decentralized membership service) to reduce the average node degree while guaranteeing (with high probability) that the events posted on each topic can be routed solely through the nodes interested in this topic (in other words, the overlay is topic-connected). SpiderCast is unique in maintaining an overlay topology that scales well with the average number of topics a node is subscribed to, assuming the subscriptions are correlated insofar as found in most typical workloads. Furthermore, the degree grows logarithmically in the total number of topics, and slowly *decreases* as the number of nodes increases.

We show experimentally that, for many practical workloads, the SpiderCast overlays are both topic-connected and have a low per-topic diameter while requiring each node to maintain a low average number of connections. These properties are satisfied even in very large settings involving up to $10,000$ nodes, $1,000$ topics, and 70 subscriptions per-node, and under high churn rates. In addition, our results demonstrate that, in a large setting, the average node degree in SpiderCast is at least 45% smaller than in other overlays typically used to support decentralized pub/sub communication (such as e.g., similarity-based, rings-based, and random overlays).

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems[Distributed applications]

Keywords

Pub/sub systems, overlay networks, scalability, peer-to-peer

DEBS '07, June 20–22, 2007 Toronto, Ontario, Canada

1. INTRODUCTION

Publish/subscribe (pub/sub) is a popular paradigm for supporting many-to-many communication in a distributed system. In a *topic-based* pub/sub system, messages (or *events*) are published on abstract event channels called topics. Users interested to receive messages published on certain topics issue *subscribe* requests specifying their topics of interest. The pub/sub infrastructure then guarantees to distribute each newly published event to all the users that have expressed interest in the event's topic. Due to its simple interface and decoupling of publishers and subscribers, pub/sub-based middleware is commonly used to support many-to-many communication in a wide variety of applications, such as enterprize application-integration [6], stock-market monitoring engines [21], RSS feeds [15], on-line gaming, and many others.

Consider a large data center with thousands of nodes offering a large variety of application services that are accessed through the Internet. Each node hosts dozens of applications and each application is replicated at many nodes for performance and availability, thereby creating overlapping multicast domains each of which is allocated a topic with most nodes being both publishers and subscribers. Furthermore, the deployment of applications on nodes may be dynamic and dependent upon the relative load incurred upon the nodes.

Since both centralized and fully-meshed communication will clearly not scale in this deployment scenario, our goal is to build a decentralized infrastructure in which the nodes are first, dynamically organized into an application level *overlay network*, and the resulting network is then used for the event routing. In this paper, we focus on constructing overlay networks whose topological properties allow for efficient topic-based event routing. Specifically, we present *SpiderCast*, a distributed overlay network construction protocol, that aims at constructing an overlay graph G satisfying the following:

(1) *Topic connectivity* — the subgraph G_t induced by the nodes interested in any topic t is a connected component. This reduces routing and communication overheads because nodes that are not interested in a certain topic need not take part in routing messages of that topic. In addition, the complexity of the routing algorithm is greatly reduced because it is much easier to build a spanning tree on G_t, when the topic induced-subgraph is connected, than a Steiner-tree over G spanning the topic (when the induced subgraph is not connected).

(2) *Scalable average node degree.* Since the cost of main-

taining overlay edges can be quite high, the average number of connections maintained by each nodes to other nodes should be as low as possible. In particular, for realiability reasons, maintaining each edge incurs the cost of periodic hearbeats and keep-alive state (or alternatively, the cost of mainitaing the connection state, if a connection-oriented transport, such as TCP, is used). In addition, reduced number of edges allows messages posted on different topics, to be aggregated into a single compound message, thereby ammortizing the header cost that can otherwise be quite significant for small messages.

(3) *Scalable topic diameter* — the diameter of the topic induced subgraph G_t should be small, in order to achieve low event propagation latency.

(4) *Churn resistance* — topic connectivity should be maintained despite nodes leaving, joining or failing. This provides resistance to node- and topic-churn in dynamic environments.

The most commonly used scheme for achieving topic connectivity is to maintain a separate overlay for each topic [3, 8]. Although such topologies scale well with the number of nodes, they, nevertheless suffer from being non-scalable with the number of subscriptions per-node. Constructing a simple logical topology (such as a tree or a ring) requires each node to maintain an average of two links per subscription. It is possible to reduce the average degree by collapsing duplicate edges, however, when the per-topic overlays are built independently, this generally has limited effect, as we will demonstrate in Section 3.9.2. Thus, with these topologies, the number of links required grows linearly with the number of topics each node is interested in. For large-scale settings, such as the one described above, or a stock-market broker interested in many dozens or even hundreds of quotes, this approach becomes impractical.

The main insight that allows SpiderCast to do better than the overlay-per-topic approach in terms of average node degree, is the fact that under typical workloads there is substantial correlation between the interests of different nodes [15, 21]. While overlay-per-topic approaches may exploit this correlation in a post-processing edge-collapsing stage, SpiderCast builds a single overlay that takes into account this correlation in the process of choosing neighbors.

The SpiderCast protocol is fully distributed. Each SpiderCast node has some knowledge on the interests of other nodes in the system. Using this knowledge, each node tries to locally determine to which nodes to connect in order to achieve topic connectivity, while keeping node degree as small as possible.

The inspiration for the protocol comes from a random graph argument [23]: In order to achieve topic connectivity, it is sufficient that for every node v, and for every topic t in which v is interested, v has random links to a small number k of other nodes interested in t. Overlay networks built according to this design principle exhibit robust connectivity, diameter logarithmic in the number of nodes, and churn resistance, for k values as small as three [16]. The innovation in SpiderCast is the efficient use of this principle in a topic-based pub/sub setting.

Thus, the rationale behind the local neighbor selection criteria for each node is to ensure that each topic in the node's interest is *K-covered*, i.e., represented in the subscription of at least K of its neighbors. The main challenge and contribution of SpiderCast is in covering all the node's interests with a small total number of links while guaranteeing that a sufficient number of those links are chosen at random.

To this end, the neighbor selection algorithm in SpiderCast combines two heuristics, *greedy* and *random* coverage, that can be seen as inducing two separate overlays in parallel and merging them at the end. The algorithm is parameterized by two integers K_g and K_r and attempts to achieve K_g greedy coverage and K_r random coverage for every topic of interest. Both heuristics are applied incrementally starting from an empty set of neighbor links, adding one link at a time, and maintaining a set S of topics that are not yet *K-covered*. The greedy coverage heuristic selects as neighbor the node that covers the maximal number of topics from S. In contrast, the random coverage heuristic randomly selects a node whose addition as a neighbor would cover at least one topic from S. The exact values chosen for K_g and K_r express the tradeoff between the average node degree, and topic connectivity. It is therefore desirable to have the nodes to choose *most* of their neighbors in the greedy fashion, and only a *few* neighbors (ideally, none at all) randomly.

In this paper, we investigate SpiderCast's performance in terms of topic-connectivity, average node degree, topic-diameter and resistance to churn. We do so by using a simulated settings involving up to $10,000$ peers, hundreds of topics, and tens of subscriptions per-peer. Our evaluation shows that with a proper combination of the greedy and random coverage heuristics, SpiderCast constructs a low-degree overlay in which all the nodes interested in a given topic form a connected component. Surprisingly, in many practical workloads, it is possible to achieve connectivity for all topics by choosing $K_g = 3$ and $K_r = 0$ (i.e., with the greedy coverage links alone). We also find that the topic-diameter grows logarithmically with the number of subscribers per-topic. According to our evaluation, the average node degree grows sub-linearly with the number of topics and *decreases* as the number of nodes increases. Most importantly, we show that the node degree scales well with the number of subscriptions per-node. Our experiments indicate that the node degree is evenly balanced across the nodes. Simulations with a realistic churn model indicate that SpiderCast maintains its topic-connectivity and low degree despite high churn rates, displaying good resistance to churn.

Finally, we compare SpiderCast with other techniques of building an overlay for topic-based pub/sub. Specifically, we compare SpiderCast with a ring-per-topic overlay, an overlay constructed based on a similarity heuristic [9], and a fully random overlay. Our results show that, in a large setting, the average node degree of the SpiderCast overlays is at least 45% smaller than the average degree of those overlay building methods.

Contributions. In summary, this paper makes the following contributions:

- It introduces a distributed protocol, called SpiderCast, based on two novel heuristics, namely the greedy and random coverage, that exploits similarity in the individual subscriptions in order to construct a low-degree topic-connected overlay for pub/sub communication. The construction is *adaptive* to the subscription similarity, and in typical (correlated) workloads a node maintains substantially *less* links that its subscription size (see Figure 8).
- It describes a distributed protocol that constructs a low-degree topic-connected overlay while requiring each

node to know the identities and interests of only 5% of the total number of nodes.

- Finally, it features a thorough evaluation of SpiderCast in very large settings involving up to 10,000 nodes, 1,000 topics, and 70 subscriptions per-node, and under high churn rates.

2. THE SPIDERCAST OVERLAY

2.1 Overview of the Protocol

The SpiderCast overlay protocol has two main components: the membership protocol (see Section 2.2), and the overlay construction and maintenance protocol (see Section 2.3). Both protocols are fully distributed. The construction protocol aims to achieve connectivity and low diameter for the entire set of topics, while maintaining as few overlay links as possible.

We define the *interest* of a node to be the list of topics to which the node has either subscribed to, or is going to publish on. The neighbor selection algorithm in SpiderCast combines two *local* heuristics, *greedy* and *random coverage*. In both heuristics, each node n tries to *cover* K times each of the topics in which it is interested. That is, for each topic t in which n is interested, n tries to maintain connections to K other nodes that are also interested in topic t. The random and greedy coverage heuristics differ in the way they choose the next neighbor, and are assigned different coverage parameters — K_g and K_r, respectively. The greedy coverage heuristic selects a neighbor that minimizes the number of topics which are not yet K_g covered. In contrast, the random coverage heuristic *randomly* selects a node whose addition as a neighbor would reduce the number of topics that are not yet K_r covered.

According to theory of k-regular random graphs[1], for $K_r \geq 3$, if each node achieves K_r (random) coverage, then for each topic t, all the nodes interested in topic t form a connected component (with high probability) whose diameter grows logarithmically with the number of subscribers to this topic [16, 23]. While such a coverage heuristic achieves the desired connectivity and low diameter per each topic, it does not exploit correlated workloads, which are common in practice, e.g., in pub/sub applications as RSS [15] and stock-market monitoring engines [21]. This is where the greedy coverage heuristic comes into play. As we show in Section 3, in many practical workloads, each link created by the greedy heuristic covers on average much more than a single topic, whereas each link created by the random coverage heuristic covers only about a single topic. In principle, however, greedy coverage alone does not ensure (with high probability) the desired topic connectivity. Thus, the exact values chosen for K_g and K_r express the tradeoff between the average node degree, and interest-based connectivity.

2.2 The Membership Service

Both the greedy and the random coverage heuristics require each node to maintain an *interest view* of other nodes in the system. The *interest view* includes the identities of other nodes along with their *interests*, and may be partial and randomized. In Section 3.7, we experimentally show that such an interest view can be readily implemented by distributed probabilistic membership protocols, such as [1, 11], augmented with the interest information. Specifically, we experimentally show that it is sufficient for each node to know the identities and interests of only 5% of the nodes in order to achieve both low average node degree and topic-based connectivity. For the rest of this paper, we therefore assume that such a service exists, and will not provide further details of its implementation.

[1]Random graphs in which each node has exactly k neighbors

2.3 Building and Maintaining the Overlay

Both the greedy and random neighbor maintenance tasks execute exactly the same code with the exception of the neighbor selection routine. In addition, each of these two tasks independently manipulates its own set of the data structure consisting of precisely the same collection of variables. We therefore describe the implementation of only one of them without an explicit reference to the exact type of neighbors being maintained. We use K to refer to the coverage parameter.

The data structures maintained by each node p are shown in Figure 1. The most important part of it is the *neighbors* set that for each current neighbor q of p, holds q's identifier (id), q's degree, q's target degree (see below), and q's current interest. In addition, each node holds its own interest in *self_interest*.

We assume the existence of a standard failure detection mechanism based on heartbeats. The heartbeats are also used to periodically update a node's neighbors with some elements of its internal state, such as its degree, and target degree.

The neighbor maintenance task starts from an empty *neighbors* set, and incrementally adds neighbors. Neighbors are added (according to the greedy or random heuristic) until the node reaches K-coverage, that is, each topic in the node's interest is represented by the interests of at least K of its neighbors. However, a node will not add neighbors without a limit. The number of neighbors is limited to be below $L^{max} + Margin$ (where $Margin$ is a small constant, e.g., 5). When the degree exceeds L^{max}, a node will stop adding new neighbors, and actively try to disconnect from some of its neighbors. Note that L^{max} is chosen to be $K \cdot |self\_interest|$, so that in the worst case, a node reaches K-coverage with each neighbor covering a single topic out of *self_interest*. In most cases, and especially with the greedy heuristic, most nodes reach K-coverage with less then L^{max} neighbors, because each neighbor typically covers more than one topic, on average.

Nodes are added into the $neighbors$ set by either *sending* connect requests, or by *accepting* connect requests. It is therefore possible for a node to become *over-covered* — that is, some neighbors may be removed from the $neighbors$ set without hampering the K-coverage property of the node. Thus, whenever a node becomes *over-covered*, it will try to disconnect from from some existing neighbors whose removal would not affect the desired coverage level of the $self\_interest$. This, however, must be done carefully. It may be possible for node p to remove node q from its $neighbors$ set and stay K-covered, whereas q would loose its K-coverage as a results of this disconnection (or q may had been under-covered to begin with). In order to address this issue, the $neighbors$ set is augmented with the degree and target degree of each neighbor. This allows each node to deduce the coverage state of its neighbors, as will be explained in the sequel.

Parameters:
- K: the desired interest coverage
- $Margin$: The number of additional links the node is allowed to maintain after the desired interest coverage has been reached
- L^{max}: an upper bound on L
- *algorithm_version*: greedy or random neighbor selection

Data structure:
- *id*: this node's identifier
- *interest*: a set of topic-id's
- *self_interest*: the *interest* of this node
- *interest_view*: a set of pairs $\langle id, interest\rangle$
- *neighbors*: sets of records $\langle id, degree, target, interest\rangle$, initially $\emptyset$
- *connect_cand_from_redirect*: a set of node identifiers, initially $\emptyset$
- L: the (adaptive) target number of neighbors

Figure 1: Data Structure and Parameters used by the Neighbor Maintenance Implementation

The neighbor maintenance task is composed of two main parts. The CONNECT routine tries to obtain K-coverage by connecting to some new neighbors. It does so until K-coverage is achieved, or until L^{max} is exceeded. The DISCONNECT routine aims to keep the node's degree from growing too much, by trying to disconnect from some existing neighbors whose removal would not hamper the desired coverage level of *self_interest*. It does so whenever the node's degree exceeds L^{max}, or when the node is *over-covered*.

The following section provides a detailed description of the neighbor maintenance implementation.

2.3.1 The Neighbor Maintenance Implementation

The neighbor maintenance task implementation is comprised of several routines (see Figures 2,3), and message and event handlers (see Figure 4). The main routine, called MAINTAINNEIGHBORS, appears in Figure 2. Its goal is to maintain K-coverage of the node's *self_interest* with as few neighbors as possible, but with less than $L^{max} + Margin$ neighbors. It executes in an infinite loop, as long as $K > 0$ (i.e., it is possible to disable the greedy or random maintenance routine by setting $K = 0$).

Whenever MAINTAINNEIGHBORS starts a new iteration of the loop, it first invokes the CALCKUNCOVERED routine (see Figure 3) to calculate the number of topics that are not sufficiently covered by the current node neighbors' interest (Figure 2, line 5). Based on the CALCKUNCOVERED's return value, either one of the CONNECT or DISCONNECT routines gets invoked. If some insufficiently covered (i.e., *under-covered*) topics are found, and the overall number of the node's neighbors is smaller than L^{max}, the CONNECT routine will try to add a new neighbor (Figure 2, lines 7–10). If, on the other hand, there are no under-covered topics or, the node's degree is higher than L^{max}, the DISCONNECT routine will try to remove a neighbor (Figure 2, line 12). These two routines are described in more detail below.

Connect: The CONNECT procedure tries to establish a connection with a new peer. The new peer to connect to is either chosen by the NEXTCOVERAGENODE routine, or from among the peers accumulated in the *connect_cand_from_redirect* set. The set *connect_cand_from_redirect* contains the *id*s of nodes that this node was redirected to, after trying to connect to some node that could accept it as neighbor (more on redirect and its rationale in the following). The NEXTCOVERAGENODE routine is substituted by either NEXTGREEDY-

```
1.  procedure MAINTAINNEIGHBORS():
2.      gap ← 0
3.      loop as long as K > 0
4.          L^max ← K · |self_interest|
5.          under_covered ← CALCKUNCOVERED(
↩                              self_interest,neighbors)
6.          if (under_covered > 0) then
7.              L ← L^max
8.              gap ← L − |neighbors|
9.              if (gap > 0) then
10.                 CONNECT()
11.         if (under_covered = 0)  ∨
↩                 (under_covered > 0 ∧ gap < 0) then
12.             DISCONNECT()
13.         sleep(connect_timeout)

14. procedure CONNECT()
15.     if connect_cand_from_redirect = ∅ then
16.         n ← NEXTCOVERAGENODE(algorithm_version)[a]
17.     else
18.         n ← some node from connect_cand_from_redirect
19.         remove n from connect_cand_from_redirect
20.     send ⟨CONNECT, |neighbors|, L, self_interest⟩ to n

21. procedure DISCONNECT()
22.     L ← min(L^max, |neighbors|)
23.     over ← |neighbors| − L
24.     m ← DISCONNECTCANDIDATE()
25.     if (m ≠ ⊥ ∧ over > 0) then
26.         send ⟨DISCONNECT⟩ to m
27.     else if (m ≠ ⊥)
28.         under_covered ← CALCKUNCOVERED(
↩                              self_interest,neighbors − {m})
29.         if (under_covered = 0) then
30.             send ⟨DISCONNECT⟩ to m
```

[a]Either GREEDY or RANDOM, see Fig 5.

Figure 2: The Neighbor Maintenance Routines

```
1.  function int CALCKUNCOVERED(interest,neighbors)
2.      u ← 0
3.      for each topic ∈ interest do
4.          cover ← {node ∈ neighbors : topic ∈ node.interest}
5.          if (|cover| < K) then
6.              u ← u + 1
7.      return u

8.  function node DISCONNECTCANDIDATE()
9.      high_degree_neighbors
↩                 ← {n ∈ neighbors : n.degree > n.target}
10.     cands ← ∅
11.     u_min ← ∞
12.     for each n ∈ high_degree_neighbors do
13.         u ← CALCKUNCOVERED(
↩                              self_interest,{neighbors − n})
14.         if (u < u_min) then
15.             cands ← {n}
16.             u_min ← u
17.         else if (u = u_min) then
18.             cands ← cands ∪ {n}
19.     if (cands ≠ ∅) then
20.         return random member of cands
21.     else
22.         return ⊥
```

Figure 3: Auxiliary routines

COVERAGENODE for the greedy neighbor maintenance, or NEXTRANDOMCOVERAGENODE for the random neighbor maintenance (see Figure 5).

When a node n receives a CONNECT request (see Figure 4 lines 1–9), it accepts it if its degree is lower than $L^{max} + Margin$. In this case, the requesting node is added

```
1.  upon receive ⟨CONNECT, degree, target, interest⟩
↩                                                  from n do
2.      if (|neighbors| < L^max + Margin) then
3.          addConnection(n,degree,target,interest)
4.          if (L < L^max) ∧ (|neighbors| < L + Margin) then
5.              L ← L + 1
6.      else
7.          cands ← {p ∈ neighbors : p.degree < p.target
↩                                                  +Margin}
8.          m ← argmax_{p∈cands}{|p.interest ∩ n.interest|},
↩                                  with ties broken randomly[a]
9.          send ⟨REDIRECT, m⟩ to n

10. upon receive ⟨REDIRECT, m, interest⟩ from n do
11.     connect_cand_from_redirect
↩                       ← connect_cand_from_redirect ∪ {m}

12. upon receive ⟨CONNECT-OK, degree, target, interest⟩
↩                                                  from n do
13.     if (|neighbors| < L^max + Margin) then
14.         neighbors ← neighbors ∪ {n, degree, target, interest}
15.         if (L < L^max) ∧ (|neighbors| < L + Margin) then
16.             L ← L + 1
17.     else
18.         send ⟨LEAVE, n⟩

19. upon receive ⟨LEAVE⟩ from n do
20.     removeConnection(n)

21. upon receive ⟨DISCONNECT⟩ from n do
22.     under_covered ← calcKUncovered(
↩                               self_interest, neighbors − {n})
23.     if (|neighbors| > L ∨ under_covered = 0) then
28.         removeConnection(n)
24.         send ⟨DISCONNECT-OK⟩ to n
25.         if under_covered = 0 then
26.             L ← |neighbors|

27. upon receive ⟨DISCONNECT-OK⟩ from n do
28.     removeConnection(n)

29. upon failureDetectionSuspect(node n) do
30.     removeConnection(n)

31. procedure addConnection(
↩            node n, int degree, int target, set interest)
32.     neighbors ← neighbors ∪ {n, degree, target, interest}
33.     send ⟨CONNECT-OK, |neighbors|, L, self_interest⟩

34. procedure removeConnection(node n)
35.     remove n from neighbors
36.     under_covered ←calcKUncovered(
↩                               self_interest, neighbors)
37.     if (|neighbors| < L) ∨ (under_covered > 0) then
38.         wake up connectivity task
```

[a] $\text{argmax}_{x\in X}\{f(x)\}$ returns $x^* \in X$, so that $\forall x \in X$, $f(x^*) \geq f(x)$

Figure 4: Message and failure detection event handlers

to the *neighbors* set and a CONNECT-OK message is sent. Otherwise, the node redirects the requesting node (by issuing a REDIRECT message) to a node $m \in neighbors$ such that the following holds: (1) m has not reached its target degree, and (2) m shares the maximum amount of interest with the requesting node. Whenever a node n receives a REDIRECT message (see Figure 4 lines 10–11), it will add its sender to the *connect_cand_from_redirect* set. In turn, n will try to connect to that node at the next iteration of the MAINTAINNEIGHBORS routine.

The process of adding links continues until all the topics

```
1.  function node nextGreedyCoverageNode()
2.      cands ← interest_view − neighbors
3.      uncovered ← ∅
4.      for each topic ∈ interest do
5.          cover ← {node ∈ cands : topic ∈ node.interest}
6.          if (|cover| < K_g) then
7.              uncovered ← uncovered ∪ {topic}
8.      node ← argmax_{n∈cands}{|n.interest ∩ uncovered|},
↩                                  with ties broken randomly
9.      return node

10. function node nextRandomCoverageNode()
11.     cands ← interest_view − neighbors
12.     uncovered ← ∅
13.     for each topic ∈ interest do
14.         cover ← {node ∈ cands : topic ∈ node.interest}
15.         if (|cover| < K_r) then
16.             uncovered ← uncovered ∪ {topic}
17.     cands ← {n ∈ {interest_view − neighbors_r} :
↩                                  |n.interest ∩ uncovered| > 0}
18.     node ← a random member of cands
19.     return node
```

Figure 5: The greedy and random neighbor selection routines

are K-covered for the first time, or until the node's degree reaches the upper bound L^{max}, in which case, the node will not try to initiate new connections with new peers anymore.

Disconnect: The DISCONNECT routine starts with the node setting its adaptive degree target L to the minimum of L^{max} and $|neighbors|$ (Figure 2, lines 22–26) thus indicating that the node has reached (or exceeded) the minimum degree required to K-cover its entire set of topics. The values of $|neighbors|$ and L are included in the CONNECT and CONNECT-OK messages, and are periodically distributed to the node's neighbors, piggy-backed on the heartbeat messages. Those values are stored in the *degree* and *target* fields of the respective *neighbors* set entry. This way, the neighbors of a fully-K-covered node q will know whether it has reached K-coverage or not (based on whether $q.target \leq q.degree$ or not).

The node then invokes DISCONNECTCANDIDATE (Figure 3, lines 8–22) to select a neighbor to disconnect from. This neighbor is chosen from among those whose degree has exceeded the minimum degree required for the complete coverage of their interest (line 9), and whose removal would have the minimum impact on the K-coverage of the node's interest (lines 12–18). If such a candidate neighbor is found, it will be sent a DISCONNECT request if one of two conditions apply: (1) the node's degree is above L^{max}, or (2) the candidate can be removed without causing this node to be under-covered. This ensures that (1) the degree will not grow much above L^{max}, and that (2) over-covered nodes try to reduce their degree to minimum in which they are still K-covered.

Whenever a node p receives a DISCONNECT request from another node q (Figure 4 lines 21–26), it will disconnect from q if it can remove q without causing its interest to become under-covered, or if it has more than L^{max} neighbors. If indeed p decides to disconnect from q, it will send the DISCONNECT-OK message which will cause q to remove p from its *neighbors* set.

Redirect: The REDIRECT messages are necessary to prevent a case in which a node p tries to connect to same node q again and again and is being rejected. This is the reason why the nodes in *connect_cand_from_redirect* are given priority when choosing the next node to connect to (Figure 2,

line 15–19). Note that neighbors that have exceeded their target degree will never be chosen as redirect candidates. A complementary mechanism (not presented in the pseudo-code) is to quarantine the nodes to which recent CONNECT requests where sent for a specified period of time.

2.4 Handling dynamic changes

The failure detection event handler, shown in Figure 4, lines 29–30, simply removes the suspected node from the *neighbors*. Subsequently, the suspect node's neighbors will try to connect to new neighbors at the next round of the neighbor maintenance task. An orderly leave involves sending the LEAVE message to all the neighbors and has essentially the same effect. Whenever a node p changes its interest, the change is propagated through the membership service and via the heart beat messages. The neighbors of p, and in fact, any other node in the overlay will take this change into account in the next round of the neighbor maintenance task.

3. EVALUATION

In this section, we evaluate the performance of SpiderCast. We have implemented the code of SpiderCast in Java and simulated it using the DESMO-J discrete-event simulator[2]. As in other studies [17, 22], most of our simulations are static, i.e., all the nodes are created simultaneously and remain up throughout the experiment. In Section 3.8, we also consider dynamic simulations, in which nodes join and leave the SpiderCast overlay. In most of our experiments, we assume that the identity and the subscription list of other nodes is available to each node, i.e., full membership service. In Section 3.7, however, we also consider simulations with a partial membership service. Throughout this section, we use (x, y)-coverage to denote SpiderCast protocol with K_g=x and K_r=y.

3.1 Workload models

In each static SpiderCast simulation, both the number of topics and the number of nodes are fixed throughout the simulation. We run simulations with the number of topics ranging from 100 to 1000, and number of nodes ranging from 1000 to $10,000$. Each node is subscribed to T topics, where T is chosen uniformly at random from the interval $[min : max]$.

A given topic t_i is chosen with a probability p_i, where $\sum_i p_i = 1$. The value of p_i is distributed according to some random distribution of topic popularity. In most of our experiments, we use a Zipf distribution with the α exponent set to 0.5, i.e., $p_i \propto \frac{1}{i^{0.5}}$. Our choice of topic popularity distribution is based on a recent study [15] of the RSS pub/sub system that shows that this distribution faithfully describes the feed popularity distribution. We henceforth refer to this distribution simply as RSS distribution. In Section 3.3 and Section 3.4, we also consider other distributions of topic popularity such as exponential, uniform, and a Zipf distribution with the α exponent set to 2. In the exponential distribution we use, the probability to choose one of the 10% most popular topics is 0.55. This distribution was used in [21] to study stock popularity in the New York Stock Exchange (NYSE). We henceforth refer to this distribution simply as NYSE distribution. We also refer to a distribution in which most of the nodes are subscribed to almost the identical set of topics as a *heavily skewed* distribution. An example of a heavily skewed distribution is a Zipf distribution with $\alpha = 2$.

As in other studies, e.g., [17], in most of our experiments all the nodes are subscribed to a fixed number of topics, i.e., min=max. In Section 3.6, we show that for a given average number of subscriptions per node, the values of min and max have a negligible effect on the average node degree.

[2]http://asi-www.informatik.uni-hamburg.de/desmoj/

3.2 Evaluation Criteria

We evaluate SpiderCast according to following criteria: (1) topic connectivity, (2) average node degree, (3) topic diameter, and (4) churn resistance.

Topic connectivity and average node degree are complementary measures of communication overhead (see Section 1). Topic diameter is defined as the maximum hop-count between any two nodes, on the same topic-induced subgraph. It represents the maximum event propagation delay under the assumption that the overlay is built in a local network with uniform between-node propagation delays. This assumption is typically true for the data-center example given in Section 1. We defer the treatment of what happens when this assumption is not met to Section 5. We measure churn resistance by verifying that the overlay remains topic-connected, and maintains low average node degree, despite nodes leaving, joining, and changing their interest.

3.3 Topic Connectivity and Parameter Setting

We ran a large number of experiments, using the workloads described in Section 3.1, with varying values of K_g and K_r, and check topic connectivity. It turns out that in all the topic popularity distributions except one, setting $K_g = 3$ and $K_r = 0$ was sufficient to guarantee topic connectivity, with the lowest average number of links. The distributions for which the $(3, 0)$-coverage was sufficient were the RSS, NYSE, and Uniform distributions. Let us emphasize that those are the distributions characterizing many practical workloads.

The topic popularity distribution for which the $(3, 0)$-coverage setting was not enough to ensure topic-connectivity was the heavily-skewed Zipf($\alpha = 2$) distribution. Setting $K_g = 3$ and $K_r = 1$ ensured topic connectivity, at the cost of increasing average node degree.

In order to demonstrate the impact of different settings of K_g and K_r, let us consider the Zipf($\alpha = 2$) distribution, with each node interested in 10 topics out of a total of 100 topics.

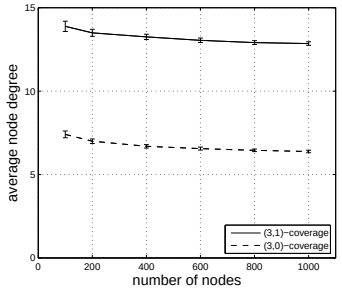
With such a heavily skewed distribution, most of the nodes are interested in the 10 most popular topics. That is, the interests of the different nodes are very similar. As a result, the $(3, 0)$-coverage setting will produce an overlay with a low average node degree. Specifically, many nodes will have a degree of only 3, and the average node degree will be lower than the average number of subscriptions per-node.

Indeed, as Table 6(a) shows, in an experiment with 6000, 8000, and $10,000$ nodes, there are 15, 14, and 14 partitioned topics, respectively. We note, however, that even with a heavily skewed distribution of topic popularity and low number of subscriptions per-node, the percentage of partitioned topics is small ($\leq$15%).

In order to eliminate topic partitions in such settings, each node could increment either K_g or K_r, or both. Note however, that based on the K-regular random graph argument

#nodes	1K	2K	4K	6K	8K	10K
#partitioned topics	0	0	0	15	14	14

a) The number of topic partitions, Zipf distribution with α= 2, K_g=3, K_r=0



(b) average node degree.

Figure 6: Zipf distribution with α=2, 10 subscriptions per-node.

(see Section 2), increasing K_g alone will have a little impact on connectivity. Indeed, Setting $K_g = 4$, $K_r = 0$ does not achieve topic connectivity in this case, whereas $K_g = 3$, $K_r = 1$ achieves topic connectivity.

Note though, that improving connectivity by increasing K_r, comes on the account of increasing the average node degree. However, as can be seen in Figure 6(b), increasing K_r from 0 to 1, results in only a small increase in the average node degree (from $\sim$7 to $\sim$13).

These results can be interpreted in the following way. In the RSS, NYSE, and Unifrom distributions, the interests of different nodes are sufficiently random so that the $(3, 0)$-coverage setting is producing a quasi-random overlay topology that achieve topic connectivity, in accordance with the random graph argument. In contrast, in the Zipf($\alpha = 2$) distribution there is not enough randomness among the node interests for the $(3, 0)$-coverage setting to produce topic connectivity. In that case, randomness has to be introduced intentionally, by means of the random coverage heuristic.

3.3.1 Adaptive Parameter Setting

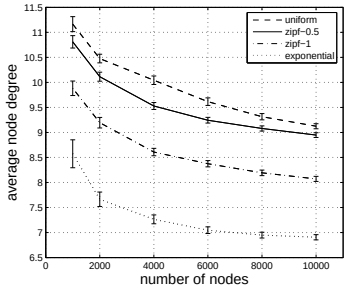
Based on the above observations, we propose an adaptive scheme of setting the parameters K_g and K_r which is as follows: at each node n, K_g is always set to 3, and K_r is initially set to 0. After achieving K_g-greedy-coverage, n checks its degree. If it is equal to K_g, then n increments its K_r parameter, and tries to randomly cover all the topics in which it is interested. After each increment of K_r (and an additional random coverage), n checks if its degree is lower than the number of topics to which it is subscribed. If its degree is lower than its number of subscriptions and $K_r \leq 3$, then n further increments its K_r parameter trying to achieve an additional random coverage. Otherwise, K_r is not further increased.

We verified that the adaptive parameter setting strategy indeed works for all the workloads that we previously presented. Note that apart from the Zipf($\alpha = 2$) distribution, all other distributions settle with $(3, 0)$-coverage. Hence, the rest of the results in this section are reported for the runs with $K_g = 3$ and $K_r = 0$.

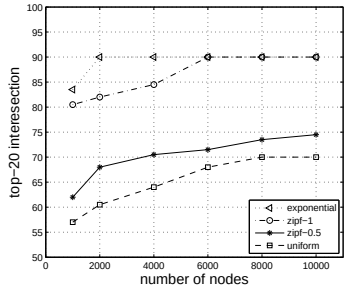
3.4 Effect of Topic Popularity Distribution

In this section, we study the effect of the topic popularity distribution on the topic connectivity, the average node degree, and the maximum topic diameter. We use four distributions: i) RSS distribution; ii) a Zipf distribution with the $\alpha = 1$; iii) NYSE distribution; and iv) a Uniform distribution. In all of the experiments in this section there are 100 topics and each node is subscribed to 10 topics. We varied the number of nodes from 1000 to $10,000$.

Interestingly, in all of the experiments, all the topics are connected (i.e., all the nodes that are interested a given topic form a connected component), although we use (3,0)-coverage. This is because none of these distributions is heavily skewed. Figure 7(a) depicts the average node degree and the 95% confidence intervals around the average degree for each experiment. As the figure shows, regardless of the topic popularity distribution, (3,0)-coverage achieves efficient aggregation: in all of our experiments, the average node degree is less than 12. This means that, on average, each node succeeds to cover each of the ten topics to which it is subscribed with less than 12 links. In addition, in all of our experiments, the 95% confidence intervals are small. This means that SpiderCast distributes the load fairly among all nodes. We also measure the maximum topic diameter for each distribution. In experiments with the Uniform, RSS, a Zipf($\alpha = 1$), and NYSE distributions, the maximum topic diameter was 6, 6, 7, and 8, respectively.



(a) average node degree



(b) top-20 intersection

Figure 7: (3,0)-coverage for different topic popularity distributions.

Finally, we note that the lowest and highest average node degree is achieved in experiments with the NYSE distribution and the uniform distribution, respectively. In addition, with the four distributions, the average node degree decreases with the number of nodes.

To explain these results, we will first introduce the following metric that captures the degree of similarity in the individual node subscriptions exhibited by a given workload: define the *intersection* (in %) between the subscription lists

of two nodes, n and m, as $100 \cdot \frac{|n.interest \cap m.interest|}{\min(|n.interest|, |m.interest|)}$. Let *top-d-intersection*$_n(d)$ ($tdi_n(d)$) be the average intersection between the subscription list of n and the subscription lists of the d nodes that have the highest intersection value with respect to n; and *top-d-intersection(d)* ($tdi(d)$) be $\frac{\sum_{n \in nodes} tdi_n(d)}{|nodes|}$. We note that the value of $tdi(d)$ is dependent on the overall number of nodes and the number of subscriptions per-node: increasing the number of nodes also increases value of $tdi(d)$, and (until a certain threshold is reached) increasing number of subscriptions per-node reduces $tdi(d)$ (see Section 3.5.3).

Given this metric, it is not unreasonable to hypothesize that with (3,0)-coverage, a high $tdi(d)$ value implies a low average node degree, since a node can greedily cover all the topics to which it is subscribed with a small number of links. To test this conjecture, we calculate the value of $tdi(20)$ for each distribution (see Figure 7(b)). We set d to 20 since with experiments with a uniform distribution several nodes can have the degree of roughly 20. Indeed, as Figure 7(b) shows, there is a strong correlation between the value of $tdi(20)$ and the average node degree: for each pair of distributions, d_1 and d_2, if the $tdi(20)$ value with d_1 is higher than the $tdi(20)$ value with d_2, then the average node degree in experiments with d_1 is lower than the average node degree in experiments with d_2.

3.5 Scalability

We now examine the scalability of SpiderCast with the number of nodes, topics, and subscriptions per-node. Throughout this section, a node's subscription list is distributed according to the RSS distribution.

3.5.1 Scalability with the Number of Nodes

In this section, we run experiments in which each node is subscribed to only 10 topics, and we increase the number of nodes from 1000 to 10,000, and the number of topics from 100 to 200. Figure 8(a) depicts the average node degree and the 95% confidence intervals around the average degree for each experiment. As Figure 8(a) shows, for a given number of topics, the average node degree *decreases* as the number of nodes grows. And for a given number of nodes, increasing the number of topics moderately increases the average node degree, suggesting a good scalability with the number of topics. We explain these results in Figure 8(d), in which, for each experiment, we calculate the $tdi(20)$ value. As Figure 8(a) and Figure 8(b) show, increasing the number of nodes increases the $tdi(20)$ value, and hence also reduces the average node degree. And increasing the number of topics reduces the $tdi(20)$ value, and hence also increases the average node degree.

We also measure the maximum node degree in each experiment. In an experiment with 100 and 200 topics, the maximum node degree is 32 and 31, respectively. While the maximum node degree can be roughly 2 or 3 times the average node degree, the 95% confidence intervals around the average degree depicted in Figure 8(a) show that most of the nodes have roughly the same degree. This means that SpiderCast fairly distributes the load among all the nodes.

Finally, we report about the maximum topic diameter. In an experiment with 100 and 200 topics, the maximal topic diameter is 6, in both cases. This means that a topic diameter grows logarithmically with the number of subscribers to the topic.

3.5.2 Scalability with the Number of Topics

In order to study the scalability of SpiderCast's overlay to the number of topics, we ran experiments in which the number of nodes is fixed to 10,000 and each node is subscribed to 20 topics. In these experiments, we increase the number of topics from 100 to 1000. Figure 8(b) and Figure 8(e) depicts the average node degree and the $tdi(40)$ value for each experiment, respectively.

As figure 8(b) shows, increasing the number of topics results in a moderate increase in the average node degree, suggesting good scalability with the number of topics. Again, these results are well explained in figure 8(e): increasing the number of topics reduces the $tdi(40)$ value, and hence increases the average node degree. Increasing the number of topics also reduces the maximum topic diameter from 5 in an experiment with 100 topics to 4 in an experiment with 1000 topics. This is because increasing the number of topics also reduces the average number of subscriptions per-topic, and hence also reduces the maximum diameter.

3.5.3 Scalability with the Number of Subscriptions Per-Node

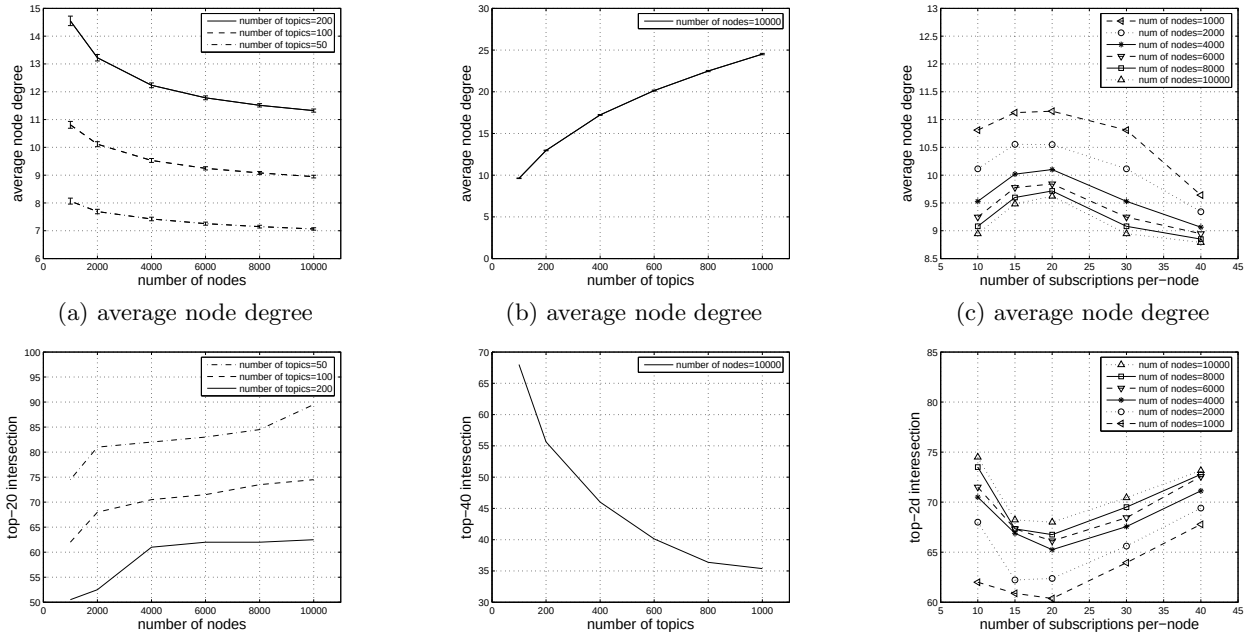
In order to study the scalability of SpiderCast's overlay to an increasing number of subscriptions per-node we ran experiments with 100 topics and 10, 15, 20, and 40 subscriptions per-node. We also varied the number of nodes from 1000 to 10,000. Figure 8(c) depicts the average node degree for each experiment. Figure 8(f) depicts the $tdi(2d)$ intersection for an experiment with d subscriptions per-node.

As Figure 8(c) shows, increasing the number of subscriptions per-node from 10 to 15 and then from 15 to 20 increases the average node degree, since each node needs to cover more topics. However, increasing the number of subscriptions per-node from 20 to 30 and then from 30 to 40 *decreases* the average node degree. This phenomena is easily explained in Figure 8(f). As this figure shows, increasing the number of subscriptions per-node from 10 to 15 and then from 15 to 20 decreases the $tdi(2d)$ value, since, until a certain threshold, increasing the number of subscriptions per-node reduces the $tdi(2d)$ value. However, above this threshold, which is around 20 in this setting, increasing the number of subscriptions per-node *increases* the $tdi(2d)$ value, since each node is subscribed to many topics, and hence the overlapping between two nodes' subscription lists is large.

3.6 Non-Equal Subscription List Sizes

We now consider a setting in which the number of subscriptions is varied across different nodes. In the first and second experiments, the size of a given node's subscription list is chosen uniformly at random from the interval $[10:30]$ and $[10:70]$, respectively. Hence, in these experiments, the average number of subscriptions per-node is 20 and 40, respectively. We compare the average node degree in these experiments with the average node degree in experiments in which each node is subscribed to 20 and 40 topics. Figure 9(a) and Figure 9(b) show the node degree histogram for experiments with an average node degree of 20 and 40, respectively.

These results indicate that for a given *average* number of subscriptions per-node, the actual distribution of the number of subscriptions per-node has a negligible effect on the *average node degree*. In addition, we note that in all the experiments, the histogram of the node degree is concentrated



(a) average node degree (b) average node degree (c) average node degree

(d) $tdi(20)$ (e) $tdi(40)$ (f) $tdi(2d)$ intersection

Figure 8: Scalability of SpiderCast to the number of nodes, topics, and subscriptions per-node.

around the average node degree. This implies that a node's degree is proportional to the number topics to which the node is subscribed.

3.7 The Effect of Partial Membership Views

In all of the experiments reported above, each node knows the identity and interest of every other node (full membership service). In this section, we report on simulations with a membership service providing incomplete membership views. In these simulations, upon bootstrap, each node's membership view contains the identities and interests of $p\%$ of the nodes chosen uniformly at random of a set that contains all the nodes. Such a membership service models a lightweight randomized membership service, e.g., Lpbcast [11]. We run two sets of simulations with 4,000 and 8,000 nodes, and we varied the value of p from 5% to 100%. In all of the experiments, each node is subscribed to 10 topics, and a node's subscription list is distributed according to the RSS distribution. Remarkably, in all of our experiments, all the constructed overlays achieve topic connectivity even when each node knew the identities and interests of only 5% of the total number of nodes. Figure 10 depicts the average node degree as a function of p. As the figure shows, the size of each node's membership view has a small effect on the average node degree. This implies that SpiderCast can use a lightweight randomized membership service, which makes SpiderCast a practical P2P protocol.

3.8 Dynamic Setting

In this section, we report on simulations in which nodes dynamically join and leave the overlay. As opposed to static simulations, in a dynamic simulation each node alternates between being connected and disconnected from the SpiderCast overlay. Each time a node wakes up, it remains connected for a time interval that is distributed exponentially with an expectation of λ. And when it disconnects, it remains disconnected for a time interval that is distributed exponentially with an expectation of $\frac{\lambda}{4}$. Every time a node (re-)joins the overlay, it does so with a different interest. We run two sets of simulations with 1,250 and 2,500 nodes. Thus, at any given time, in the first and second set of simulations an average of $1,000$ and $2,000$ of the nodes are connected to the overlay, respectively. The simulation were run assuming full membership.

We varied the value of λ from 100 to 500 seconds, and ran the simulation for $1,000$ seconds. Every 200 seconds, we took a snapshot of the overlay, and analyzed the connectivity of each topic as well as the average node degree. In a snapshot taken at time t, we excluded the nodes that joined the overlay in the interval $[t-2, t]$ seconds. Remarkably, in all the simulations and all the snapshots, all the topics were connected regardless of the value of λ, i.e., the churn rate. In addition, as Figure 11 shows, the average node degree was also not affected by the churn rate. This implies that SpiderCast is most suitable for being deployed over a dynamic setting in which nodes leave, join, an change their interest.

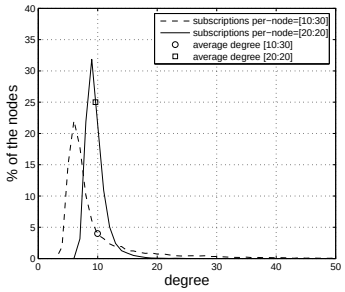
3.8.1 Join/Leave Message Overhead

Finally, we measure the average number of control messages associated with a join/leave operation. In experiments with $1,000$ and $2,000$ nodes, the average number of control messages associated with a join/leave operation was 38 and 34, respectively.

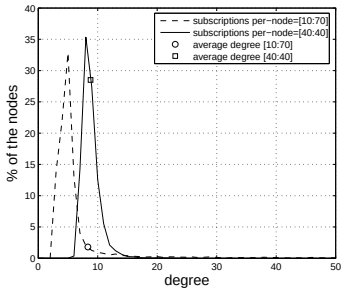
If we divide these numbers by the average node degree (13 and 12, resp.), we get that the average overhead of creating a link is approximately 3 control messages.

If we divide these numbers by the size of each node's subscription list (10), we get that the average overhead of covering a single topic is less than 4 control messages.

3.9 Other Overlay Construction Methods



(a) an average of 20 subscriptions per-node.



(b) an average of 40 subscriptions per-node.

Figure 9: The effect of the distribution of the number of subscriptions per-node on the average node degree.

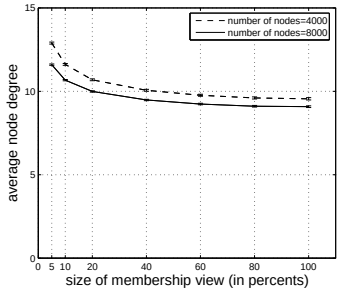


Figure 10: The effect of the size of the membership view on the average node degree.

In Section 3.9.1, we compare SpiderCast to two unstructured overlays, and in Section 3.9.2, we compare SpiderCast to a protocol that constructs a structured ring per-topic.

3.9.1 Comparison with similarity-based and fully random overlays

We now compare SpiderCast to two unstructured overlays: an overlay based on the similarity heuristic [9] and a random overlay. We also compare SpiderCast to overlays in which part of the links are random ones and the rest of the links are created according to the similarity heuristic. In order to allow a fair comparison between the overlays, we use SpiderCast's code for simulating all the overlays with the exception of replacing the functions for choosing neighbors nextGreedyCoverageNode and nextRandomCoverageNode with the functions nextSimilarityNode and nextRandomNode. Each invocation of nextSimilarityNode returns a node n from the current node's interest view so that the intersection between the current node's subscription list and n's subscription list is the highest among all the nodes in the current node's interest view. Each invocation of nextRandomNode return a random node from the current node's membership view. A similarity(x, y) simulation creates for each node x links based on the similarity heuristic and y random links. We note that if x=0 (and y>0), then the constructed overlay is a random overlay. We denote such an overlay as random(y). In order to allow a fair comparison, we use a full membership view for all the implementations. In all the simulations, the topic popularity distribution is distributed according to RSS distribution, and each node is subscribed to 10 topics out of 100 topics. In each set of simulations, we varied the number of nodes from 1000 to $10,000$.

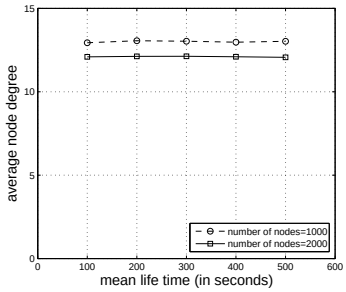


Figure 11: The effect of the mean life time (churn rate) on the average node degree.

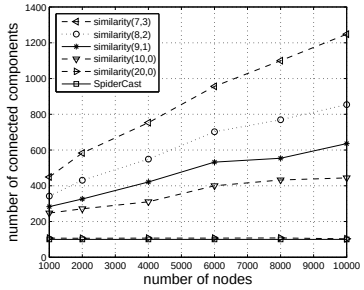
We first run a set of SpiderCast simulations in which the average node degree is between 8.95 and 10.81. Next, we run four sets of simulations with the similarity-based implementation: i) similarity$(10, 0)$; ii) similarity$(9, 1)$; iii) similarity$(8, 2)$; and iv) similarity$(7, 3)$. Next, we run random(10) simulations. In all of these six sets of simulations, the average node degree is roughly the same. Finally, we run a set of simulations with similarity$(20, 0)$, in which the average node degree is twice the average node degree of either of the above simulations. In Table 12, we report about the average node degree in each experiment.

protocol/ number of nodes	1000	2000	4000	6000	8000	10000
SpiderCast ($(3, 0)$-coverage)	10.81	10.11	9.53	9.24	9.08	8.95
similarity(10,0)	10.08	10.07	10.08	10.09	10.08	10.08
similarity(9,1)	10.22	10.22	10.20	10.22	10.21	10.22
similarity(8,2)	10.20	10.19	10.19	10.20	10.20	10.20
similarity(7,3)	10.21	10.21	10.20	10.22	10.21	10.21
random(10)	10.82	10.54	10.34	10.12	10.08	10.11
similarity(20,0)	20.04	20.04	20.04	20.04	20.03	20.04

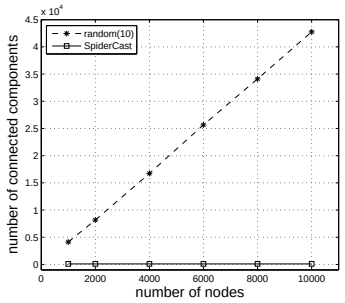
Figure 12: Average degree for a given number of nodes.

In order to evaluate the communication overhead incurred by each overlay, we calculate the number connected components for each simulation. The communication overhead is minimized when each topic is connected, i.e., in a simulation with 100 connected components (recall that the overall number of topics is 100). Figure 13(a) and Figure 13(b) depict the number of connected components for each experiment. In all the SpiderCast experiments there are 100 connected components, i.e., *all* the topics are connected, and hence the communication overhead is minimal. In contrast, in *each* of the experiments with the similarity heuristic, or the random

heuristic, or any combination of these two heuristics, the number of connected components is larger than 100. Even with experiments with the similarity heuristic with twice the average node degree the number of connected components is larger than 100. Moreover, in all the simulations with the similarity heuristic (and the random heuristic), the number of connected components *increases* with the number of nodes. Hence, as opposed to SpiderCast, the similarity heuristic cannot scalably support a large number of nodes.



(a) SpiderCast vs a similarity-based overlay.



(b) SpiderCast vs a random overlay.

Figure 13: SpiderCast vs a similarity-based and random overlays of the same average degree.

Finally, we note that, with the similarity heuristic, creating random links on the expense of similarity-based links increases the number of connected components. In contrast, recall that in SpiderCast simulations with a heavily skewed distribution, adding random coverage links reduces the number of connected components. This is since SpiderCast tries to randomly *cover* the topics to which the current node's is subscribed, whereas in simulations with the similarity-based overlay random links are added regardless of the topics to which the current node is subscribed. Hence, a random coverage is much more efficient than simply adding random links.

3.9.2 Comparison with ring-per topic protocol

To see that exploiting interest correlation could indeed result in a substantial scalability improvement, we now compare the average node degrees of the overlays created by SpiderCast and by a *Ring-Per-Topic (RingPT)* protocol (similar to [22]). In the RingPT protocol, the per-topic connectivity is achieved by maintaining a separate logical ring, ordered by the node identifiers, for each topic. Additionally, in the RingPT protocol, we merge duplicated links. In all the simulations in this section, the topic popularity distribution is distributed according to RSS distribution, and each node is subscribed to 10 topics out of 100 topics.

As Figure 14 shows, the average node degree in the experiments with RingPT is between 51% and 82% higher than the average node degree in the experiments with SpiderCast. Moreover, whereas the average node degree in experiments with RingPT does node change with the number of nodes, in the experiments with SpiderCast, the average node degree *decreases* with the number of nodes. These results shows that SpiderCast achieves substantially higher scalability than a naive protocol that does not exploits correlated workloads.

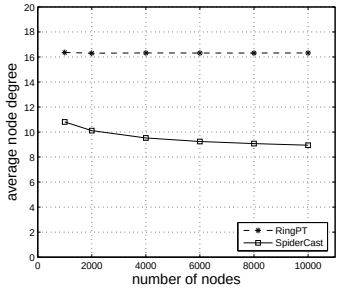


Figure 14: SpiderCast vs a ring per-topic protocol.

4. RELATED WORK

Many traditional implementations of pub/sub (see e.g., [5, 2]) rely on centralized components, such as message brokers, and/or fixed topologies and therefore, offer limited scalability in large and dynamic settings. This motivated interest in decentralized pub/sub solutions, and in particular, those based on P2P architectures.

In order to achieve selective event dissemination, most existing P2P pub/sub systems leverage the properties provided by structured overlay networks [4, 20], and/or organizing peers into global dissemination topologies, such as multicast trees [8]. A smaller number of pub/sub architectures are based on unstructured overlays: Sub-2-Sub [22] that employs a combination of an unstructured overlay and additional ring structures to support content-based pub/sub. It is assumed in [3] that topics are organized into a hierarchy in the naming space and constructs a hierarchy of unstructured overlays that is based on it. Although relying on structured elements is instrumental for routing efficiency, maintaining global topologies incurs the cost of reconfiguration in the presences of dynamic changes, thus making these systems less favorable in highly dynamic settings [10]. Other P2P-based implementations of content-based pub/sub, such as DPS [2] and GosSkip [12], focus on efficient algorithms for content filtering which are less relevant for topic-based pub/sub systems, such as SpiderCast.

The impact of the links based on similarity in the nodes' subscription on the performance of the event filtering in content-based P2P pub/sub systems was investigated in [9]. This work provides a valuable insight that the similarity-based links can indeed be useful for more efficient event dissemination in overlay-based pub/sub systems. However, as we show in Section 3.9, the links based on the similarity alone are insufficient for efficient event dissemination as the communication cost incurred by propagating events on less popular topics can be quite high.

There exist a few optimization techniques that mitigate the problem of large fanout in the overlay-per-interest approach but do not solve it completely. Topic (or, more gen-

erally, interest) aggregation into groups increases dissemination and filtering overhead because a single connected component for a topic group may translate into multiple components for each individual topic. Constructing a hierarchy of overlays based on interest containment [3, 9, 2] precludes fully unstructured solutions and does not solve the problem when subscriptions exhibit strong similarity without being contained in each other.

Using pub/sub communication for supporting distribution of active web content was investigated in [17, 18]. Both these systems mainly focus on efficient cooperative polling of the publishers by the subscribers, and are less concerned with maintaining specific overlay topologies, which is our focus in this paper.

5. DISCUSSION

We have presented SpiderCast, an overlay-based infrastructure for scalable topic-based pub/sub. SpiderCast scales well with the number of nodes, number of topics, and the size of subscriptions as long as the latter are correlated, as shown via thorough performance evaluation. In the future, we are going to apply our k-coverage-based construction technique to content-based pub-sub by generalizing the coverage to be measured in units of the event space rather than in the number of topics.

The fact that SpiderCast attains per-topic connectivity with high probability in both static and dynamic settings can be leveraged for efficient event dissemination. Specifically, it is possible to build a dissemination topology for each topic in a distributed fashion (e.g., a distributed tree using commonly known algorithms such as the one used in [13]) utilizing only Spidercast-created overlay links. The challenge of this approach is to take advantage of the low node fan-out in order to maintain routing state that scales well with the number of node's interests.

In order provide resilience to churn, it is also necessary to dynamically detect failures and disconnections and reconcile the topology. The following two approaches can be used to make dissemination over per-topic topologies reliable in presence of churn: systems like Overcast [13] buffer and retransmit messages whereas Pbcast [7] and Bullet [14] employ tree-based best effort dissemination and periodic communication with siblings in order to detect and retrieve missing messages. The difference between these approaches represents a well-known tradeoff: while retransmission over the tree is more efficient in terms of communication overhead, methods such as gossiping are more robust and allow for faster completion of missing messages.

The idea of enhancing randomly constructed unstructured overlays with links based on geographical proximity in order to meet requirements for reduced latency was introduced in [16, 19]. Although we made the simplifying assumption of uniform between-node delays, let us note that the SpiderCast protocol can be easily augmented to incorporate the said techniques.

6. REFERENCES

[1] A. Allavena, A. Demers, and J. Hopcroft. Correctness of a gossip based membership protocol. In *PODC*, 2005.

[2] E. Anceaume, M. Gradinariu, A. K. Datta, G. Simon, and A. Virgillito. A semantic overlay for self-* peer-to-peer publish/subscribe. In *ICDCS*, 2006.

[3] S. Baehni, P. T. Eugster, and E. Guerraoui. Data-aware multicast. In *DSN*, 2004.

[4] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-Based Publish-Subscribe over Structured Overlay Networks. In *ICDCS*, 2005.

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

[6] K. Birman. *Building Secure and Reliable Network Applications*. Manning, 1996.

[7] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

[8] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: a large-scale and decentralized application-level multicast infrastructure. *IEEE J. Sel. Areas in Comm.*, 20(8):1489–1499, 2002.

[9] R. Chand and P. Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In *Euro-Par 2005 Parallel Processing, LNCS*, volume 3648, pages 1194–1204. Springer Verlag, 2005.

[10] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM*, 2003.

[11] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003.

[12] R. Guerraoui, S. Handurukande, and A.-M. Kermarrec. Gosskip: a gossip-based structured overlay network for efficient content-based filtering. Technical Report IC/2004/95, EPFL, Lausanne, 2004.

[13] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and H. W. O. Jr. Overcast: reliable multicasting with an overlay network. In *OSDI*, 2000.

[14] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *SOSP*, pages 282–297, October 2003.

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

[16] R. Melamed and I. Keidar. Araneola: A Scalable Reliable Multicast System for Dynamic Environments. In *NCA*, 2004.

[17] V. Ramasubramanian, R. Peterson, and E. G. Sirer. Corona: A high performance publish-subscribe system for the world wide web. In *NSDI*, 2006.

[18] D. Sandler, A. Mislove, A. Post, and P. Druschel. Feedtree: Sharing web micronews with peer-to-peer event notification. In *IPTPS*, 2005.

[19] C. Tang, R. N. Chang, and C. Ward. Gocast: Gossip-enhanced overlay multicast for fast and dependable group communication. In *DSN*, 2005.

[20] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *DEBS*, 2003.

[21] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In *17th IASTED Int'l Conf. Parallel and Distributed Computing and Systems*, 2005.

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

[23] N. Wormald. Models of random regular graphs. *Surveys in Combinatorics*, 276:239–298, 1999.