PolderCast: Fast, Robust, and Scalable Architecture for P2P Topic-Based Pub/Sub

Vinay Setty, Maarten Van Steen, Roman Vitenberg, Spyros Voulgaris

PolderCast: Fast, Robust, and Scalable Architecture for P2P Topic-based Pub/Sub

Vinay Setty[1], Maarten van Steen[2], Roman Vitenberg[1], and Spyros Voulgaris[2]

[1] Department of Informatics, University of Oslo, Norway
{vinay,romanvi}@ifi.uio.no
[2] Department of Computer Science, VU University, Amsterdam, The Netherlands
{steen,spyros}@cs.vu.nl

Abstract. We propose PolderCast, a P2P topic-based Pub/Sub system that is (a) fault-tolerant and robust, (b) scalable w.r.t the number of nodes interested in a topic and number of topics that nodes are interested in, and (c) fast in terms of dissemination latency while (d) attaining a low communication overhead. This combination of properties is provided by an implementation that blends deterministic propagation over maintained rings with probabilistic dissemination following a limited number of random shortcuts. The rings are constructed and maintained using gossiping techniques. The random shortcuts are provided by two distinct peer-sampling services: Cyclon generates purely random links while Vicinity produces interest-induced random links.
We analyze PolderCast and survey it in the context of existing approaches. We evaluate PolderCast experimentally using real-world workloads from Twitter and Facebook traces. We use widely renowned Scribe [5] as a baseline in a number of experiments. Robustness with respect to node churn is evaluated through traces from the Skype superpeer network. We show that the experimental results corroborate all of the above properties in settings of up to 10K nodes, 10K topics, and 5K topics per-node.

Keywords: Publish/Subscribe, Peer-to-Peer, Gossiping

1 Introduction

Publish/subscribe (pub/sub) has become a popular communication paradigm that provides a loosely coupled form of interaction among many publishing data sources and many subscribing data sinks [8]. Many applications report benefits from using this form of interaction, such as application integration [20], financial data dissemination [2], RSS feed distribution and filtering [15], and business process management [14]. As a result, many industry standards have adopted pub/sub as part of their interfaces. Examples of such standards included WS Notifications, WS Eventing, OMG's Real-time Data Dissemination Service, and the Active Message Queuing Protocol.

In pub/sub, subscribers convey their interests in receiving messages and publishers disseminate publication messages. The language and data model to subscribe and publish vary among systems. In this paper, we focus on the topic-based pub/sub model. In a topic-based system, publication messages are associated with topics and subscribers register their interests in receiving all messages published to topics of interest. While traditional pub/sub implementations are either centralized or based on a federated organization of cooperatively managed servers, an increasingly higher number of pub/sub applications are being deployed in P2P environments [22]. Following this trend, a number of decentralized topic-based pub/sub systems have been proposed over the last decade [3,5,7,9,16,19,27,28]. These systems build a decentralized infrastructure in which the nodes are first dynamically organized into an application-level overlay network, and the resulting network is subsequently used for event routing.

The designers of these systems are facing an uphill struggle because of the distinctively high number of desirable characteristics that a large-scale P2P pub/sub system has to possess all at once in order to be a viable practical solution. In particular, the list includes: (1) Correct delivery of all publications, i.e., absence of false negatives or deterministic 100% hit-ratio guarantee in a failure-free run, (2) *High hit-ratio* under realistic node churn, (3) Fast recovery at the end of a churn period and mending of the overlay so as to achieve 100% hit-ratio, (4) *Low degree* of overlay nodes, (5) *Relay-free routing* (also called topic-connectivity), which means that only subscribers interested in a topic are involved in routing events for that topic, (6) *Scalability* with the number of nodes, topics, number of nodes interested in a topic, and number of topics a node is interested in, (7) Effective dissemination: *fast*, with as *little duplicate delivery* as possible, and *fair distribution of load* due to routing and processing, and (8) *Low overhead* of overlay maintenance. The design challenge is amplified due to a number of trade-offs: low node degree and relay-free routing, robustness under churn and lack of duplicate delivery, scalability and precise delivery with few false negatives and false positives are fundamentally at odds with each other. Furthermore, each of the principal solution approaches provides a bundle of desirable and undesirable properties at the same time: dissemination over multicast trees is fast and without duplication but it is fragile, whereas gossiping is robust but lacking deterministic delivery guarantees.

In this paper, we present PolderCast[1], a P2P architecture for topic-based pub/sub. To the best of our knowledge, PolderCast is the first solution that takes all of the above factors into account and harmonizes them. In order to substantiate this claim, we present a survey of existing approaches and analyze their performance with respect to most of the above characteristics.

This combination of desirable properties is provided by an implementation that blends deterministic propagation over maintained rings with probabilistic dissemination following a limited number of carefully selected random shortcuts. Per-topic rings allow for relay-free routing and 100% hit-ratio in absence of node

[1] The term is inspired by the Dutch *polder model*, in which diverse societal groups collaboratively negotiate to obtain broadly supported solutions.

churn, yet they are constructed in such a fashion so as to reuse the same links for multiple rings thereby minimizing the average node degree. Although at a conceptual level this overlay structure encompasses a separate Hybrid Dissemination [25] overlay *per-topic*, our design leverages interest locality to produce a single composite overlay with substantially fewer links and hence, lower node degrees. Our implementation is based on a new efficient epidemic-based algorithm for creating and maintaining the proposed overlay in a self-organizing way.

We evaluate and validate the properties of our system using extensive simulations in large-scale settings of up to 10K nodes, 10K topics, and 5K topics per-node. We use real-world traces from Twitter and Facebook social networks to model subscriptions. Robustness with respect to node churn is evaluated through traces from the Skype super-peer network. We empirically show that our system (1) converges fast, (2) provides 100% hit-ratio in the absence of node churn and reasonably good hit-ratio in the presence of node churn, (3) has logarithmic dissemination speed in terms of number of hops and (4) has constant factor traffic overhead. We use Scribe [5] as a baseline in a number of our experiments.

2 Preliminaries

The system consists of a set $\mathcal{V}$ of nodes. Each node in the system has a *unique* identifier (e.g., a hash of its IP address), assigned to it when joining the system. Node identifiers are assumed to be sortable and to occupy a circular value space. We assume that the underlying communication network is fully connected, in the sense that *any* node can send a message to *any* other node, provided it knows its IP address.

The topic-based publish/subscribe communication system is organized around a set $\mathcal{T}$ of topics. Each node can play the role of a subscriber or publisher or both. A subscriber v expresses its interest in a set of topics $\mathcal{T}_v \subseteq \mathcal{T}$. We call $|\mathcal{T}_v|$ the *subscription size* of node v. A publisher posts an event on exactly one topic t. The published event should be delivered to *all* $|\mathcal{V}_t|$ ($\mathcal{V}_t \subseteq \mathcal{V}$) subscribers interested in t (no false negatives) and *only* to them (no false positives).

Both publishers and subscribers are allowed to join and leave at any moment, without any prior notice. Node crashes are, therefore, inherently dealt with as ungraceful leaves. In fact, there is no way to distinguish between the two. We assume that a node that leaves and rejoins after a while can remember its prior state.

3 Survey of Related Approaches

In practice, a pub/sub system should satisfy a wide spectrum of desirable properties in the context of high robustness, low dissemination latency, low communication overhead, and high scalability. Many of those properties exhibit an inherent trade-off with each other so that striking the right balance is a central challenge in a pub/sub system design and a guiding objective for our approach.

Table 1. Comparison of State of the Art with PolderCast

Property\System	Scribe [5]	Vitis [19]	SpiderCast [7]	StAN [16]	daMulticast [3]	Polder-Cast
Central nodes*	RV	RV&GW	WB	None	None	None
High hit-ratio under churn?	✗, see Sec. 6.6	✓	N/A	N/A	✓	✓
100% hit-ratio in absence of churn?	✓	✓	N/A	N/A	✗	✓
TCO?	✗	✗	Prob.	Prob.	Det.	Det.
Degree of node v	$O(\log \lvert\mathcal{V}\rvert)$	$O(1)$	$O(\lvert\mathcal{T}_v\rvert)$	$O(\lvert\mathcal{T}_v\rvert)$	$\Theta(\lvert\mathcal{T}_v\rvert)$	$O(\lvert\mathcal{T}_v\rvert)$
Incl. dissemination?	✓	✓	✗	✗	✓	✓
Average Duplication Factor	None	Scoped flooding	N/A	N/A	Gossiping	$\leq$ Fanout(f)
Average Delay	$\mathrm{O}(\log \lvert\mathcal{V}\rvert)$	$\mathrm{O}(\log^2 \lvert\mathcal{V}\rvert)$	N/A	N/A	$O(\log \lvert\mathcal{V}_t\rvert)$	Typically $O(\log \lvert\mathcal{V}_t\rvert)$#

* RV: Rendezvous. GW: Gateway. WB: Weak bridge.
# For more details refer to Sec. 6.4 and the discussion below in this section.

Table 1 compares the characteristics of PolderCast with principally different approaches for P2P topic-based pub/sub systems.

With respect to robustness, a pub/sub system should ideally guarantee both 100% hit-ratio without node churn and high hit-ratio in presence of node churn. Consider that existing approaches to P2P pub/sub either utilize epidemic dissemination (daMulticast [3]), or build specialized dissemination overlays. It is well-known that while robust under churn, epidemic dissemination does not provide full reliability, even in a completely static system. On the other hand, most existing dissemination overlays for topic-based pub/sub are fragile (such as dissemination trees in Scribe [5], Magnet [9], or Bayeux [28]) or at least they rely on designated nodes whose existence is critical for correct operation of distributed matching. For example, Scribe and Vitis [19] have a dedicated rendezvous node for each topic. Additionally, Vitis builds subclusters for each topic and the communication between subclusters is handled by gateway nodes. While these systems provide a number of churn-handling mechanisms, fragility of dissemination overlays or reliance on central nodes conceptually limit the potential for high hit-ratio under churn, as we further explore in our evaluation in Sec. 6.6. SpiderCast [7] builds an unstructured overlay that strives to maximize clustering of nodes according to their interest in topics. As observed in [16], this approach may yield an overlay in which highly-connected clusters are interconnected by few links, which we call *weak bridges*. Existence of such weak bridges also impacts the robustness of the system under churn.

PolderCast combines deterministic dissemination over a ring with probabilistic dissemination similar to gossiping. The former mechanism guarantees 100% hit-ratio in a static system while the latter provides a high hit-ratio under churn. This is further corroborated by the experimental evaluation in Sec. 6.6.

Consider the characteristics of the overlay built in various existing approaches: A low number of relay nodes is instrumental in reducing the communication and processing cost of dissemination as well as propagation latency expressed by path lengths. Furthermore, guaranteed absence of relays, a.k.a. *topic-connectivity* [6], simplifies message routing mechanisms. On the other hand, fanout is a com-

mon minimization parameter in overlay design, which strongly affects system scalability.

Unfortunately, the desirable characteristics of having a low node degree and relay-free routing exhibit a fundamental trade-off [6]. At one extreme is having a fixed node degree independent of the number of topics a node is interested in. Such an approach is proposed in Vitis. This results in a relatively high number of subclusters that need to be connected by additional means, such as gateways, rendezvous nodes, and relays. Scribe builds dissemination structures on top of an underlying DHT whose node degree might be either constant or logarithmic with the total number of nodes in the system. In these systems, a pair of nodes interested in the same topic might be connected by a chain of $\Theta(\log |\mathcal{V}|)$ relays.

At the other extreme of the trade-off are systems that build and maintain a separate overlay for each topic independently, such as Tera [4] and systems that employ gossiping on a per-topic basis, such as daMulticast. These approaches guarantee topic-connectivity during stable periods without churn. However, the degree of node v in these systems is in the order of the number of subscriptions: $\Theta(|\mathcal{T}_v|)$.

SpiderCast and StAN strive to maintain a topic-connected overlay by building random links between the nodes while exploiting the correlation between node interests in order to minimize the degree. Since correlations are typically present in pub/sub workloads, this results in a lower degree compared to Tera or daMulticast. After the system becomes stable, these systems will eventually produce a topic-connected overlay with high probability. Yet, the guarantee of relay-free routing is only probabilistic, which yields low overhead and latencies, but requires additional mechanisms to route messages across potentially disconnected clusters.

The PolderCast approach we propose in this paper provides a deterministic guarantee of relay-free routing similar to Tera or daMulticast. At the same time, the degree is similar to that of SpiderCast or StAN due to exploiting correlations. As shown in Table 1, SpiderCast and StAN focus on overlay construction and maintenance and do not propose any specific routing algorithm, thereby rendering the discussion about message dissemination properties as well as hit-ratio nonapplicable to these systems.

For the rest of the approaches, we consider two salient factors that determine the efficiency of message dissemination:

(a) Average message duplication factor per node: the number of times (excluding the first) that the same published message is received by a node on average. When the routing is relay-free, average message duplication factor directly translates into the communication cost of message dissemination.

In Scribe, Magnet, and Bayeux, a routing tree is used to disseminate publications, which eliminates any duplication of messages. In the hybrid overlay approach of Vitis, the node floods a published message to those of its neighbours that are interested in the message topic. Even though Vitis has a fixed total degree per node, this fanout may be high enough so as to lead to a high number of duplicate deliveries for the same published message. In daMulticast,

the configurable fanout of the epidemic dissemination used for propagating published messages governs the duplication factor. In PolderCast there is a fixed maximum dissemination fanout f (typically f= 2) for each topic. Each node interested in the topic forwards a message only once (the first time the node receives the message) along at most f links, which gives a bound of f on the duplication factor.

(b) Average path-length: the average number of hops required for a message to reach a node interested in that message. As shown in Table 1, all of the structured and hybrid overlay approaches have an expected path length that is logarithmic or square logarithmic with the total number of nodes $|\mathcal{V}|$ in the system. Yet, the inclusion of relays nodes (both at the DHT level and pub/sub implementation level) into the dissemination path causes path lengths for some nodes being significantly longer than $O(\log|\mathcal{V}|)$, as we show in Sec. 6.4. DaMulticast performs gossiping on a per-topic basis so that the expected path length is logarithmic with the number of nodes $O(\log|\mathcal{V}_t|)$ interested in the topic.

In our approach, we also strive to achieve expected path lengths that are logarithmic with $O(\log|\mathcal{V}_t|)$ due to the random shortcuts links used for dissemination. From the results in [25], it can be derived that if there is a sufficient number (f-1) of random shortcut links between the nodes interested in a particular topic, PolderCast guarantees average dissemination path lengths for that topic to be asymptotically logarithmic. However, our dissemination mechanism uses a fixed number of random links independently of the number of topics a node is interested in. This may potentially render the dissemination mechanism ineffective for a node that is interested in many topics, in which case the average path length may become linear with $|\mathcal{V}_t|$ due to the use of ring links only. Fortunately, this scenario does not manifest itself for typical pub/sub workloads, as confirmed by the empirical results in Sec. 6. Note that the dissemination fanout f determines the base of the logarithm and as such, governs the trade-off between the dissemination speed and duplication factor.

Based on the analysis in this section, we conclude that the solution for topic-based pub/sub we propose is (a) free from rendezvous and relay nodes (b) robust and resistant to churn, and (c) it facilitates efficient message dissemination.

4 PolderCast: Disseminating Events

We present PolderCast in a top-down approach. In this section we describe the structure of the target overlay and we explain how dissemination is performed once this overlay is in place. Then, in Sec. 5, we dive into the mechanisms in charge of building and maintaining such an overlay.

4.1 The Dissemination Overlay

At a conceptual level we maintain a separate ring per topic augmented by random links shared across the topics. Each ring connects *all* subscribers of the corresponding topic and *only* them. Individual topic rings altogether form a single,

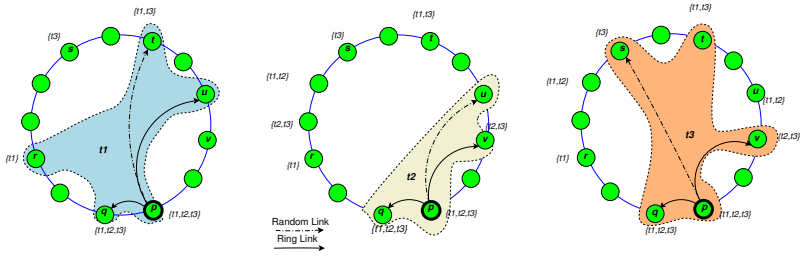

Fig. 1. Topology for three topics $\{t1, t2, t3\}$, showing the ring neighbor links and random neighbor links originating from the node p. Note that q serves as successor of p for all three topics, and v serves as predecessor of p for topics $t1, t2$ illustrating link sharing.

connected, and navigable overlay. Ensuring connectivity among all subscribers of a topic, a property known as *topic connectivity*, allows for relay-free routing among them. It is the reason why PolderCast achieves 100% hit-ratio in the absence of node churn: When an event for a certain topic reaches *any* subscriber of that topic, it is guaranteed to reach *all* remaining subscribers by being propagated along that topic's ring. While this distribution mechanism alone might be adequate for topics with a moderate number of subscribers, its linear dissemination speed does not scale with the popularity of topics. This is the reason why we introduce random links serving as dissemination shortcuts. Propagating events across (some of the) random links to arbitrary other subscribers of the same topic, accelerates dissemination to exponential speed. It additionally provides a controlled degree of redundancy that increases robustness and hit-ratio under node churn.

In this work, we request that a publisher on topic t subscribes to t prior to publishing events, thus becoming a part of the dissemination ring. This overhead for publishers is considered acceptable by most applications and in many existing pub/sub systems.

The rings for each topic are *bidirectional* and nodes are placed into rings in the order of their node ids. That is, a node p maintains, with respect to each topic t in its subscription, two links: one to its t-successor and one to its t-predecessor. The t-successor of node p is defined as the node with the *closest higher* than p's id (in modulo arithmetic), among all subscribers of topic t. The t-predecessor is defined likewise for the *closest lower* id. Fig. 1 gives a sample topology of three topics, and the respective intermingling rings.

It should be observed that while the use of rings in hybrid dissemination structures has appeared in the past [25], their application to topic-based pub/sub is new. The main challenges of using ring in pub/sub lies in combining such structures, one per topic, into a single manageable overlay. In practice, maintaining a separate ring per topic is very expensive, notably for nodes subscribed to many topics. However, it has been observed that subscriptions tend to be strongly correlated [15]. Our approach exploits this correlation in order to substantially lower the number of links maintained: A single link can serve as a ring link for multiple topics.

It is possible to build an overlay with link consolidation across the topics as the central optimization metric in mind. This approach minimizes node degree but may result in a per-topic ring being partitioned into multiple sub-rings. In order to avoid this risk, PolderCast takes a more balanced approach and builds a guaranteed ring for each topic separately but in such a way that links have a higher chance of being reused in multiple topics. Specifically, rings are constructed based on node ids instead of their subscriptions. Assume nodes p and q are both subscribed to t_1 and t_2, and they are ring neighbors for t_1. This means that they are both on the ring for t_2 and their ids are numerically close, thereby increasing the chance that they will be ring neighbors for t_2 as well. We further investigate the effect of link consolidation in our experiments in Sec. 6.

With respect to random links, their choice and quantity may have a profound impact on the performance, as discussed in Sec. 3. PolderCast combines a configurable number of random links of two types: interest-induced links formed between subscribers with similar subscriptions shorten average dissemination path lengths. At the same time, uniform random links help overcome partitions under node churn and improve load balancing by diverting incoming links from nodes that subscribe to many topics, which become a likely target for interest-induced links. We describe the algorithm for random link formation in Sec. 5 and consider the importance of the links of each type in Sec. 6.

4.2 Event Dissemination

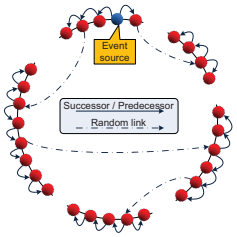

Fig. 2. Dissemination example for a particular topic, in a partitioned ring.

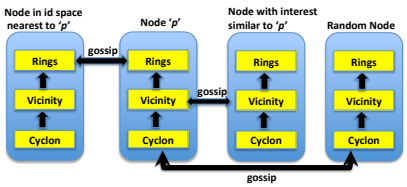

Fig. 3. Three-layered architecture. Each layer gossips with the respective layer in other nodes.

Our event dissemination protocol is inspired by that of RingCast [25] (the protocol is parameterized by a dissemination fanout, f): A node receiving an event for topic t for the *first time*, propagates it f times. Specifically, if the event has been received through the node's t-successor (or t-predecessor), it is propagated to its t-predecessor (or t-successor) and f-1 arbitrary subscribers of t. If the event was received through some third node, or if it originated at the node in question, it is propagated to both the t-successor and the t-predecessor, as well as to f-2 other subscribers of t. Finally, if a copy of this event has already been received in the past, it is simply ignored.

From the results in [25], it can be derived that if there is a sufficient number (f-1) of random shortcut links between the nodes interested in a particular topic, PolderCast guarantees average dissemination path lengths for that topic to be asymptotically logarithmic. Even under node churn PolderCast tries to achieve complete dissemination as shown experimentally in Sec. 6.6. Fig. 2 gives an intuitive illustration of dissemination in a partitioned ring.

Since we apply this dissemination protocol for multi-topic pub/sub, however, analyzing its performance in PolderCast is significantly more difficult because the random links are shared across multiple topics and the number of utilizable random links varies for each and every node. Furthermore, some of the random links are skewed towards peers with multiple overlapping topics. This may interfere with the nice property of exponential dissemination speed that is inherent to many gossiping protocols. It may also cause a node whose subscription is similar to those of many other peers to become a hotspot due to a high number of incoming random links. We evaluate these aspects experimentally in Sec. 6.

5 PolderCast: Building the Overlay

PolderCast's overlay management mechanism is built around three modules: Rings, Vicinity, and Cyclon, as shown in Fig. 3. Each module maintains its own view, managed by a separate gossiping protocol, which gossips periodically, asynchronously, and independently from the other two modules. In table below we list the parameters controlling the number of neighbors maintained (*view size*), and the maximum number of neighbors included in a gossip message (*gossip size*), per module.

module name	**view size**	**gossip size**
Rings	ℓ_{ring} (per subscribed topic)	g_{ring}
Vicinity	ℓ_{vic} (in total)	g_{vic}
Cyclon	ℓ_{cyc} (in total)	g_{cyc}

Considering a node p with topics $\mathcal{T}_p$, the three modules operate as follows. With respect to each topic $t \in \mathcal{T}_p$, the Rings module on p is responsible for discovering p's t-successor and t-predecessor. It achieves this by considering a few links to arbitrary subscribers of t as a starting point, and periodically gossiping with them to trade them for other subscribers of t of gradually closer ids.

The Vicinity module is responsible for feeding the Rings module with a few neighbors for each topic $t \in \mathcal{T}_p$, of arbitrary ids. It is based on Vicinity [23], a topology management protocol that strives at discovering for each node the *closest* other nodes based on some *proximity function*. Per the proximity function introduced in the context of PolderCast, the more topics two nodes share the closer they are ranked. Moreover, as detailed in Sec. 5.2, our proximity function dynamically adapts to favor topics currently under-represented in the Rings module.

Finally, the Cyclon module [24], is a lightweight peer sampling service [12], providing each node with a continuous stream of neighbors chosen uniformly

at random from the whole network. As detailed in Sec. 5.3, this is essential for keeping the whole overlay connected, and enabling flexible overlay maintenance in the face of failures and node churn.

For any of the three modules, node q being a *neighbor* of node p means that p has a copy of q's *profile* in the respective module's view. A node's profile contains (i) its IP address and port number, (ii) its (unique) node id, and (iii) the ids of topics the node is subscribed to, each annotated with a *priority* that node assigns to finding neighbors of that topic. The priority of a topic is determined by the number of neighbors it has in the Rings module: topics with fewer Rings neighbors are assigned higher priority. Clearly, two or more copies of a node's profile may be different, notably when the node updates its subscriptions, or reports different priorities for its topics. When gossiping to a neighbor, a node sends a fresh copy of its profile, reflecting its current state.

Note that the three gossiping protocols comprising PolderCast are executed continuously. In a network characterized by dynamicity, due to nodes departing or joining at any time, crashing, or merely changing their subscriptions, there is no notion of *final* convergence. Instead, nodes engage in a constant convergence process.

5.1 The Rings module

The Rings module manages the ring links. That is, it aims at discovering a node's successor and predecessor for each topic in its subscription, and at quickly adapting to new successors/predecessors in dynamic networks.

In that respect, each node maintains ℓ_{ring} neighbors for each topic in its subscription: $\ell_{ring}/2$ with lower and $\ell_{ring}/2$ with higher id. It periodically picks a node from its Rings view, and the two nodes exchange up to g_{ring} neighbors to help each other improve their Rings views.

Assume p selects its neighbor q for gossiping. First, p collects *all* subscribers of topics which p and q have *in common*, considering the union of views of *all three modules*. Second, it sorts them by id, and for each topic in common with q it selects the $\ell_{ring}/2$ ones with just lower and the $\ell_{ring}/2$ ones with just higher id than q's id. If more than g_{ring} nodes have been selected, it randomly picks g_{ring} of them. Finally, it sends the selected nodes (i.e., the respective node profiles) to q. Node q does the same in return.

Although the dissemination protocol requires just two ring links per topic, namely the topic successor and predecessor, Rings maintains up to ℓ_{ring} links per topic. This provides stand-by successors and predecessors to be used in case of failures or node churn. Additionally, it helps nodes navigate to their direct ring neighbors faster, once they have reached the proximity of their ids.

Finally, in order to increase the diversity of neighbors contacted for gossiping, the Rings module employs a Least Recently Used (LRU) selection policy. This prevents contacting the same neighbor twice in a short interval, when it probably has no new useful information, at the expense of not contacting some other neighbor for a much longer duration. The LRU policy also plays an important

role in churn handling by PolderCast, thus its implementation details are deferred to Sec. 5.4.

5.2 The Vicinity module

The Vicinity module is responsible for maintaining interest-induced random links, that is, randomly chosen links between nodes that share one or more topics. Such links serve as input to the Rings module, as detailed in Sec. 5.1. Additionally, they are used by the dissemination protocol to propagate events to arbitrary subscribers of a topic, as explained in Sec. 4.2.

Interest-induced random links are handled by Vicinity [23], a generic protocol for topology construction and management that lets nodes find their *closest* neighbors out of the whole network, based on some *proximity function*. In short, each node maintains a view of ℓ_{vic} neighbors and periodically gossips with them to discover nodes of even closer proximity, in which case it retains them in place of the least proximal neighbors.

Let p choose q for gossiping. Node p merges its views from all three modules. Then, it selects the g_{vic} nodes closest to q by applying the proximity function on its behalf, and ships them over to q. Upon reception, q merges the received neighbors with the union of all its views, and updates its Vicinity view to the ℓ_{vic} closest neighbors. Finally, q responds by selecting and shipping back its g_{vic} closest to p nodes.

Clearly, the proximity function plays a crucial role in Vicinity. In the context of PolderCast, the proximity function is designed to ensure that the Rings module is supplied with (arbitrary) neighbors for *all* its topics. In that respect, candidates subscribed to topics annotated with higher priority by the target node are ranked closer compared to candidates of lower priority topics. Among candidate nodes that rank equally in terms of topic priorities, proximity is determined by the number of topics shared with the target node: the more shared topics, the closer their ranking.

5.3 The Cyclon module

Uniform random links are handled by the Cyclon peer sampling service [24]. This module's purpose is twofold. First, it keeps the whole set of subscribers connected in a single partition, even in the presence of churn, large scale failures, or subscription changes. Connectivity is crucial to let new subscribers find their way to their appropriate neighborhood sets, irrespectively of where they initially joined the network. Second, it constitutes a source of links selected uniformly at random from the whole network. Such a source of random links is fundamental to the operation of the other two modules. Further details about the Cyclon protocol can be found in [24].

5.4 Churn Handling

It is a key design goal of PolderCast to provide a high hit-ratio and reasonably low delivery latency under node churn, while keeping the number of duplicate

messages controllably small. To that end, PolderCast should adapt promptly to two types of changes. First, information updates, such as newly joining nodes, new subscriptions, etc. should be propagated fast. Second, the system should quickly detect the disconnection (graceful or due to failures) of nodes, and discard related information from the network.

With respect to propagating new information fast, PolderCast relies on its fast convergence properties. When a node joins the network, for example, its Vicinity module will quickly find some neighbors for each topic. Once a neighbor has been found for some topic, the Rings module can quickly locate the appropriate successor and predecessor in an already largely connected topic ring. When a node's subscription changes, Vicinity will adjust its topic priorities to boost under-represented (new) topics. We further explore the convergence speed of PolderCast experimentally in Sec. 6.2.

With respect to ridding the system from outdated links, PolderCast employs a proactive mechanism for removing dead neighbors from node views. Whenever a node p gossips with a neighbor q, it temporarily removes q from the respective module's view, anticipating that q will respond and will be inserted anew in p's view. This way, dead neighbors are silently discarded, while alive ones are refreshed. To prevent dead neighbors from remaining indefinitely in a view, a node always selects to gossip with its least recently refreshed neighbor.

Freshness of a neighbor is approximated by an *age* field, associated with every view entry. Once per cycle, a node increments the ages of all its neighbors by one. A neighbor's age is zeroed when a gossip message (or response) is received from that neighbor. A neighbor's age is retained also when that neighbor is handed from one node to another. This way, a dead node's links will have increasingly higher chance to be selected for gossiping (and consequently discarded), even if they are copied among third nodes.

Although the age mechanism provides only an approximation of a link's freshness, it turns out to work sufficiently well for fast removal of dead links. We investigate the impact of node churn on the performance of PolderCast in Sec. 6.6.

6 Experimental Evaluation

We evaluate PolderCast by simulation based on real-world traces. We focus on the overlay properties (such as the node degree), efficiency of dissemination (delays and duplicate delivery), communication overhead of overlay maintenance, and performance under node churn (hit-ratio for message delivery and speed of convergence for overlay construction). We also compare the performance of PolderCast with Scribe [5] as a baseline.

We implement both PolderCast and Scribe using the widely adopted *PeerSim simulator* [17]. Scribe is implemented as an application atop Pastry DHT [21]. We use the implementation of Pastry for *PeerSim*, publicly available at [1]. We evaluate both PolderCast and Scribe at a scale of up to 10K nodes. Experiments of similar scale are common in this area [18, 19].

Unless otherwise mentioned, the view sizes of CYCLON and VICINITY (ℓ_{cyc} and ℓ_{vic}, respectively) were set to 20 entries each, and the gossip lengths in all three protocols (g_{cyc}, g_{vic}, and g_{ring}) were set to 10 entries. The configuration parameters for Scribe are $b = 4$ which defines the base $2^b = 16$ for the log structure of Pastry DHT and $l = 32$ for the leaves of the DHT routing table.

6.1 Experimental Settings

Subscription Workload: Our subscription workloads come from massively deployed social networks, namely Twitter and Facebook.
(1) Twitter dataset: We used a public Twitter dataset [13], containing 41.7 million distinct user profiles and 1.47 billion social followee/follower relations. In Twitter, when a user posts a message (known as a *tweet*), the tweet is delivered to all followers of that user. As such, each user is modelled as a topic and all its followers are the respective subscribers. Similarly the set of users (followees) a user Alice follows, form Alice's subscription set. Note that in Twitter, relations are unidirectional, i.e., user Alice following user Bob does not require also Bob following Alice.
(2) Facebook dataset: We used a public Facebook dataset [26], with over 3 million distinct user profiles and 28.3 million social relations as a second workload for our evaluations. Similarly to Twitter, users are modelled as topics as well as subscribers. However, in Facebook relations are bidirectional, therefore two friends in the Facebook social graph subscribe to each other in our model.

Our simulations were performed with workloads of 10K nodes (i.e., up to 10K topics and 10K subscribers), extracted from the original Twitter and Facebook social graphs in a methodology inspired from [18, 19]. More specifically, starting with a random set of a few users as seeds, we traversed the social graph using breadth first search, until the target number of nodes was reached, and *all* edges between them were extracted to our sample.

Fig. 4 shows the complementary cumulative distribution function (CCDF) of follower/followee counts for both the original Twitter(TW) and Facebook(FB) datasets, as well as for our respective extracted datasets in the inner plot. The plots indicate that the original dataset properties were retained in our extracted sample.

Publication Workload: Due to lack of publicly available real world publication workload we synthetically generate publications. We post one publication event for *each* topic, initiated by a randomly picked subscriber of that topic. Although in practice, event arrival rate may vary across different topics, we use a uniform publication rate since it has no effect on the metrics we consider in this paper.

Latency and Churn Datasets: We use the *King dataset* [11] to model communication latency between nodes. Finally, we evaluate our system under node churn, using real world churn traces: *Skype dataset*. We use Skype super-peer churn traces from [10], which tracked joining and leaving timestamps of 4000 nodes for one month, starting on September 12, 2005.

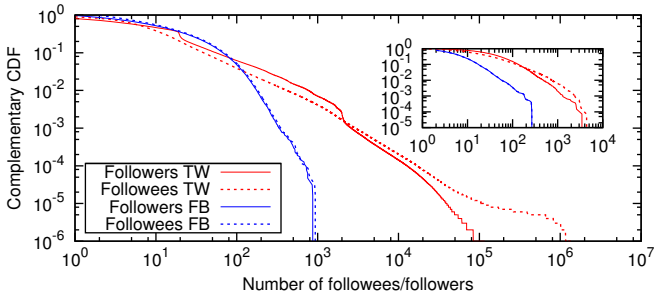

Fig. 4. Distribution of followers and followees, for the Twitter (41.7M users) and Facebook (3M users) traces. Inner plot: trace samples used (10K users).

6.2 Speed of Convergence

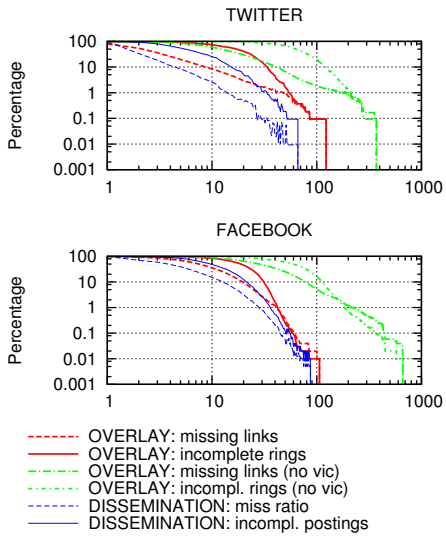

Fig. 5. Convergence speed

We first evaluate the time it takes to jump-start a PolderCast overlay from scratch. We start by 10,000 nodes that are already running Cyclon (i.e., each node has ℓ_{cyc} links to random other nodes), but whose Vicinity and Rings views are completely empty, and we let them gossip to self-organize in a PolderCast overlay. Observe that fast convergence to an optimal overlay upon the extreme case of simultaneous bootstrapping typically implies fast reconciliation after a period of milder churn.

Given the input, we start by an offline construction of correct target rings to which the systems should converge over time. Then, we deploy PolderCast.

At each cycle, we measure the percentage of target ring links that are *not* yet in place (missing links), as well as the percentage of topics for which the ring has not converged yet (incomplete rings). Fig. 5 shows these metrics for the Twitter and Facebook workloads, respectively.

In order to assess the overlay's efficiency in disseminating events, we conduct another experiment by "freezing" the overlay at the end of each cycle, and posting one event for each topic. We record the percentage of nodes that *missed* an event they should have received (miss ratio), as well as the percentage of events that did *not* make it to all subscribers of their topic (disconnected topics). These measurements are also shown in Fig. 5.

The results show that the overlay converges quite fast: Within 60 cycles, 99% of topic rings are complete. They also indicate that the PolderCast overlay is highly efficient even with partially complete rings because it takes fewer cycles

to achieve a connected overlay (0% miss ratio) per topic. This is due to propagating events across random links, provided by the combination of VICINITY and CYCLON views.

We also show that our three-layered architecture explained in Sec. Sec. 5 is essential to improve the speed of convergence. In Fig. 5 we compare the convergence speed of POLDERCAST, without the VICINITY layer in the middle, and we can see that it takes almost 3-6 times longer to converge. This is because VICINITY provides interest-induced random links, essential for speeding up the construction process.

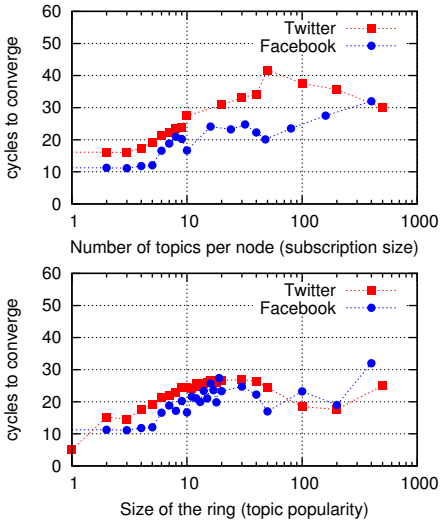

Fig. 6. Correlation between convergence speed and size of the subscription/ring

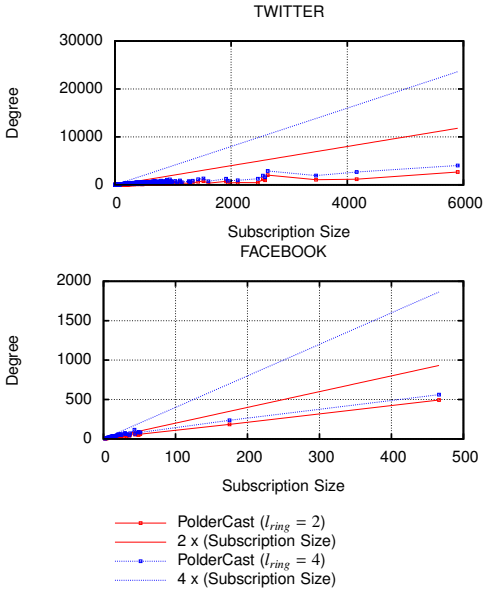

Fig. 7. Node degree in Rings layer

Apart from the speed it is also important to make sure that the overlay construction is scalable with respect to the number of nodes that participate in a ring (topic popularity) and the number of topics a node is interested in (subscription size). As shown in Fig. 6, even a node interested in over 400 topics converges reasonably fast. This is mainly due to having a higher number neighbours compared to a node interested in a few topics only, which offer it much higher reachability for a large number of topics.

6.3 Overlay Degree

In Fig. 7 we assess the effect of a node's subscription size on its RINGS view size. Due to interest locality, a single neighbor may serve multiple of its topics. This helps the node retain its RINGS outdegree low, and effectively contributes to higher scalability with respect to the subscription size of nodes. We do not

consider the degree due to random links here since their number is fixed and small compared to that of ring links.

For the Twitter data, PolderCast manages to exploit correlation in the subscriptions to a large extent. However, for Facebook data, the node degree grows almost linearly with subscription size suggesting less subscription correlation. In Scribe, the average degree of a node v in the system is bounded by the number of nodes in the Pastry routing table that point to node v. This number is logarithmic with the total number of nodes and independent of the number of topics that node is subscribed to. This may be an important advantage in the case of an extremely high number of topics a node is interested in.

6.4 Event Dissemination

We now analyze the event dissemination protocol proposed in Sec. 4.2. We measure (1) the dissemination delay, in terms of number of hops required for a publication to reach the subscribers and (2) the duplication factor, namely the ratio between the number of *all* event messages received over the number of *distinct* event messages received. The measurements were taken by injecting the publications as described earlier and averaging the two metrics for 1000 cycles. From this point on, we run PolderCast with only Facebook data with 10K nodes, omitting results for Twitter data due to lack of space.

As one can see in Fig. 8(a), with the increase in dissemination fanout the average dissemination delay significantly decreases. However, this decrease takes place at the cost of an increase in the average number of duplicate messages seen by nodes as shown in Fig. 8(b). To compare Scribe with PolderCast we plot the average delay in Fig. 8(a). We can see that the average dissemination delay in Scribe is almost 1.7 times higher than the worst-case dissemination delay of PolderCast. This is due to the long chain of nodes induced by Scribe dissemination trees, even though DHT gurantees $\log |\mathcal{V}|$ hops delay. These longer chains stem from the inclusion of relay nodes, both at the Scribe and Pastry level.

As shown in plots in Fig. 8(a,b), the choice of random shortcut links has an interesting trade-off between dissemination delay and duplicate messages. At one extreme, if we use the Cyclon view as a source for random shortcut links, neither the dissemination delay decreases, nor the duplication factor increases with the increase in fanout f. This is attributed to the fact that since the Cyclon view is limited in size, and its view is chosen in an interest-agnostic way, the random shortcuts for a topic the node is interested in are not useful for the topics of interest, forcing the dissemination protocol to fall back on ring links. On the other extreme, if we only use the Vicinity view as a source of random links, it leads to a significant decrease in average delay, at the cost of an increase in the average number of duplicates. In PolderCast we balance this trade-off by combining the Cyclon and Vicinity views, which results in the middle ground both for average delay and average duplication factor.

The choice of random shortcuts also has implications on the balancing of load on the nodes. In Fig. 8(d) one can see that if only Vicinity is used for random shortcut links, around 20% of the nodes receive messages at least 4 times. This

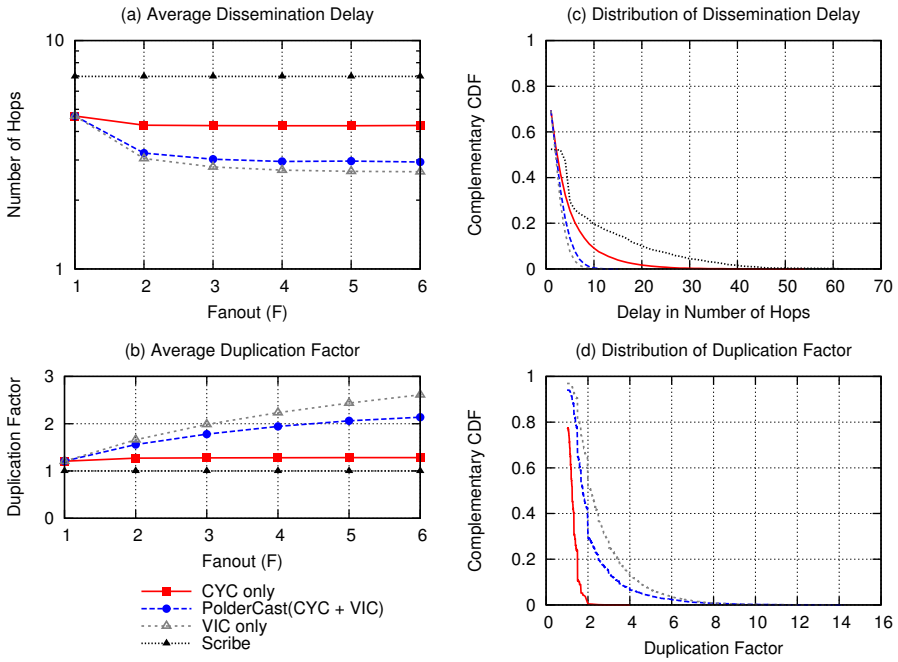

Fig. 8. Event Dissemination Analysis

is due to the fact that nodes that are interested in many topics (> 100) have a high chance to be present in the VICINITY view of many nodes. Since we use both VICINITY and CYCLON views for random shortcuts, it reduces the number of duplicate messages for nodes interested in many topics. It should be noticed that Scribe does not have any duplicate messages since messages in Scribe are disseminated using multicast trees.

In Fig. 8(c) we can see a similar pattern for dissemination delay and we again take the middle ground between the two extremes. Fig. 8(c) also shows that there is a significant number of messages in Scribe with a relatively high dissemination delay, as we explained above.

6.5 Overlay Maintenance

The next experiment aims at evaluating the overhead in overlay maintenance. We measure the number of control messages sent and received by each node to maintain the overlay. Note that as shown in Fig. 9 nodes interested in many topics (> 100) transmit a higher number of messages. This is due to the fact that they are more frequently selected as a target for gossiping. This factor does not play a significant role: the cycle duration can be chosen to be as high as 1 minute in real scenarios thereby rendering the bandwidth overhead negligible. On the other hand, more intensive control communication by nodes interested in many topics contributes to faster overlay convergence.

It is clear from Fig. 9 that Scribe incurs a higher communication overhead. The number of control messages sent and received by a node v in Scribe is

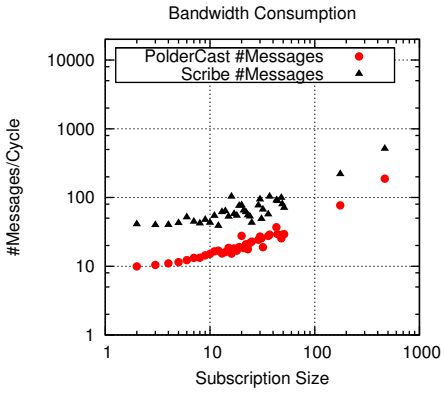

Fig. 9. Bandwith consumption

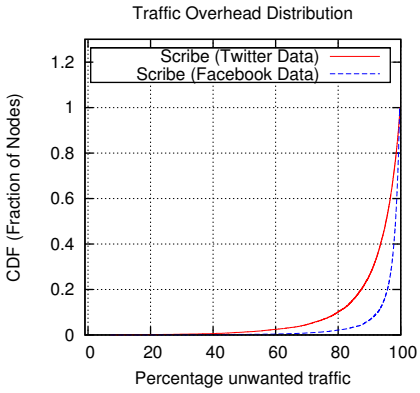

Fig. 10. Traffic Overhead

proportional to the number of subscriptions v is interested in. Even though each node has a limited number of children in the multicast tree to maintain, Scribe sends regular heartbeat messages for each topic (both topics of interest and topics for which v is a relay) to keep the trees connected.

The existence of relays and lack of topic-connectivity in Scribe additionally causes unwanted traffic passing through the nodes. We measure the amount of overall traffic (both control and application traffic) passing through each Scribe node and distinguish between the traffic relevant to the subscription topics of the node and unwanted traffic. In Fig. 10 we show the amount of unwanted traffic at each node. We can see that over 90% of the nodes receive more than 80% of unwanted traffic. Such an overhead does not exist in PolderCast since topic-connectivity ensures that each node receives only the traffic relevant to the node's subscription topics.

6.6 Message Dissemination Under Churn

In this experiment we evaluate PolderCast and Scribe publication dissemination under the churn model described earlier. We inject publications as explained earlier with fanout f set to 2. We maintain two successors and two predecessors for each topic (ℓ_{ring}= 4). To assess the resilience of our protocol to node churn, at the end of each cycle we freeze the overlay and we measure the miss-ratio, i.e., the fraction of nodes that missed at least one publication event. It is worth noting that we set the cycle duration to be 1 minute. As a consequence, we introduce 60 times more node churn during each cycle than originally provided by the churn traces. When measuring the miss-ratio, we exclude the warm-up period of 10 seconds after the node joins the network.

As shown in Fig. 11, for the Skype churn model the miss-ratio in PolderCast never grows beyond 0.01 except when there is a sharp drop in network size. In that case, the miss ratio momentarily grows to 0.04, but stabilizes quickly. This is due to (1) the use of random shortcuts, keeping the dissemination structure connected even though the ring is partitioned, and (2) since ℓ_{ring}= 4, with the failure of one successor/predecessor the ring can still stay connected. When

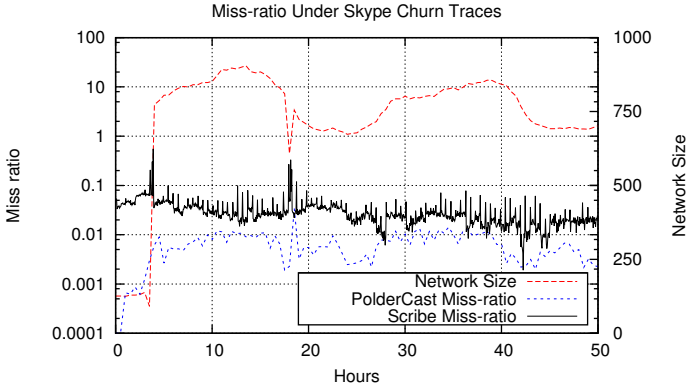

Fig. 11. Message Dissemination Under Churn

hundreds of nodes are joining the system (i.e., when there is a flash crowd), PolderCast continues to maintain the miss-ratio below 0.01.

From Fig. 11 it can be seen that Scribe has almost 10 times higher miss-ratio than PolderCast. Especially during the flash crowd at the beginning Scribe has a significantly higher miss-ratio due to a slower construction of the multicast trees when around 600 nodes join. Similarly we can see a spike in the miss-ratio when a sharp drop in network size occurs after around hours 18. There is a spike in the miss-ratio of PolderCast as well, but the relatively higher miss-ratio of Scribe is caused by the sudden departure of several rendezvous nodes.

7 Conclusions

In this paper we presented PolderCast, a P2P architecture for topic-based pub/sub which aims to achieve relay-free, fast and robust dissemination over a scalable overlay with a minimal maintenance cost. PolderCast achieves a delicate balance between these conflicting but desirable properties. We evaluated PolderCast with Scribe as baseline, using large scale simulations with publicly available real world traces from Facebook [26] and Twitter [13].

References

1. An implementation of the Pastry protocol for PeerSim. `http://peersim.sourceforge.net/code/pastry.tar.gz`.
2. Tibco rendezvous. `http://www.tibco.com`.
3. S. Baehni, P.T. Eugster, and R. Guerraoui. Data-aware multicast. In *DSN*, 2004.
4. R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *DEBS*, 2007.
5. M. Castro, P. Druschel, A.M. Kermarrec, and A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20:1489 – 1499, 2002.
6. G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *PODC*, 2007.

7. G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS*, 2007.
8. P.T. Eugster, P.A. Felber, R. Guerraoui, and A.M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35:114–131, 2003.
9. S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed. Magnet: practical subscription clustering for internet-scale Pub/Sub. In *DEBS*, 2010.
10. S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *IPTPS*, 2006.
11. K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: estimating latency between arbitrary internet end hosts. *SIGCOMM*, 2002.
12. M. Jelasity, A. Montresor, and Ö. Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321 – 2339, 2009.
13. H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW*, 2010.
14. G. Li, V. Muthusamy, and H.A. Jacobsen. A distributed service-oriented architecture for business process execution. *ACM Trans. Web*, 4:2:1–2:33, 2010.
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. M. Matos, A. Nunes, R. Oliveira, and J. Pereira. Stan: exploiting shared interests without disclosing them in gossip-based publish/subscribe. In *IPTPS*, 2010.
17. A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *P2P Computing*, 2009.
18. J.A. Patel, É. Rivière, I. Gupta, and A.M. Kermarrec. Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Computer Networks*, 53:2304 – 2320, 2009.
19. F. Rahimian, S. Girdzijauskas, A.H. Payberah, and S. Haridi. Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks. In *IPDPS*, 2011.
20. J. Reumann. GooPS: Pub/Sub at Google. Lecture & Personal Communications at EuroSys & CANOE Summer School, 2009.
21. A.I.T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. volume 2218 of *Middleware*, pages 329–350. Springer, Berlin/Heidelberg, 2001.
22. Peter Triantafillou and Ioannis Aekaterinidis. Peer-to-peer publish-subscribe systems. *Encyclopedia of Database Systems 2009: 2069-2075*, 2009.
23. S. Voulgaris. *Epidemic-Based Self-Organization in Peer-to-Peer Systems*. Phd thesis, VU Universiteit Amsterdam, 2006.
24. S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13:197–217, 2005.
25. S. Voulgaris and M. van Steen. Hybrid dissemination: adding determinism to probabilistic multicasting in large-scale P2P systems. In R. Cerqueira and R. Campbell, editors, *Middleware*, LNCS, pages 389–409. Springer, New York, 2007.
26. C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *EuroSys*, 2009.
27. B. Wong and S. Guha. Quasar: a probabilistic publish-subscribe system for social networks. In *IPTPS*, 2008.
28. S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J.D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, 2001.