OMen: Overlay Mending for Topic-based Publish/Subscribe Systems Under Churn

ABSTRACT

We propose, OMen, a distributed system for dynamically maintaining overlays for topic-based publish/subscribe (pub/sub) systems. In particular, OMen supports churn-resistant construction of *topic-connected overlays* (TCO), which organizes all nodes interested in the same topic in a directly connected dissemination sub-overlay. While aiming at pub/sub deployments in data centers, OMen internally leverages selected peer-to-peer technologies, such as T-Man as the underlying topology maintenance protocol.

Existing approaches for constructing pub/sub TCOs are (i) centralized algorithms that guarantee low node degrees at the cost of prohibitive running time and (ii) decentralized protocols that are time efficient while lacking bounds on node degrees.

We show both analytically and experimentally that OMen combines the best from both worlds. Namely, OMen achieves (i) low node degrees, close to centralized algorithms, and (ii) high efficiency, scalability, and load balance, comparable to decentralized protocols. Our evaluation uses both synthetic pub/sub workloads and real-world ones extracted from Facebook and Twitter. We generate churn traces with Google cluster data.

Keywords

Overlay, pub/sub, churn handling, topic-connected overlay, T-Man

1 Introduction

Distributed topic-based publish/subscribe (pub/sub) systems often organize nodes (e.g., brokers, servers or routers) into a federated or peer-to-peer manner as an overlay at the application or network layer. The overlay forms the foundation for distributed pub/sub and determines the performance and scalability of the system, e.g., routing complexity and message forwarding costs. Constructing the pub/sub overlay is a fundamental problem that attracts attention both in industry [2, 9, 27] and academia [5, 6, 7, 21, 26, 29].

The innate dynamism of networks requires distributed pub/sub to be capable of maintaining the properties of the overlay in presence of churn. In practice, a data center shows non-negligible variation for non-faulty running machines over time [1, 27]. Most practical pub/sub systems take into account resilience requirements in their design; for example, Yahoo! Message Broker (YMB) guarantees the delivery of published messages to all subscribers, even when facing a limited number of broker machine failures [9]. Further, the advent of new pub/sub applications, e.g., the IBM Internet of Things (IoT) foundation [2], makes it increasingly important and challenging to maintain pub/sub overlays in presence of churn.

We propose a new architecture called OMen (Overlay Mending) for maintaining dissemination overlays for topic-based pub/sub systems. The major *requirements* underlying the design of OMen are:

R1 Overlay quality:

(a) *Topic-connected overlay (TCO)*: Each topic induces a connected sub-overlay among all nodes interested in this topic [6]. In a TCO, nodes not interested in a topic never need to contribute to disseminating information on that topic. Message routing atop such pub/sub overlays saves bandwidth and computational resources otherwise wasted on forwarding and filtering out unwanted messages. A TCO also results in more efficient routing protocols, a simpler matching engine implementation, smaller forwarding tables, and higher security.

(b) *Low node degrees*: It is imperative for a pub/sub overlay to have low node degrees. While overlay designs for different applications might be principally different, they all strive to minimize node degrees, e.g., DHTs [10, 28] and small-world networks [15]. First, it costs a lot of resources to maintain adjacent links for a high-degree node (i.e., monitoring links and neighbors). For a typical pub/sub system, each link would accommodate a number of protocols, service components, message queues, etc. Second, in low-degree overlays, the set of nodes that participate in coordination of distributed tasks tends to be smaller. Particularly for pub/sub routing, the node degree directly influences the sizes of routing tables, the complexity of matching, and the efficiency of message delivery.

(c) *Small overlay diameters*: The overlay diameter impacts many performance factors for efficient routing in pub/sub, e.g., message latency. For example, the initial GooPS design required that each source-sink pair would be no more than three overlay hops apart [27].

R2 Fast recovery: It is essential to restore a TCO upon each churn event as fast as possible with minimum disruption to pub/sub communication. All desirable properties about TCOs are fragile and easily break in a dynamic environment. Fast recovery is a critical prerequisite, especially, for time-sensitive pub/sub applications, e.g., IoT [2] and stock quote notifications [30].

R3 Decentralized maintenance: It is infeasible for any node in the system to keep the global knowledge (i.e., knowledge about all the nodes in the network) and to monitor all the nodes. The handling of each simple churn event should only impact a small portion of nodes in the network.

R4 Fairness and balanced load: This refers to balancing computation, communication and storage due to overlay maintenance across all nodes in the overlay. Both restoring TCOs and monitoring the overlay operations incur computation as well as communication overhead. We need to spread this overhead fairly across all the nodes.

R5 Reliability: It is important to tolerate concurrent churn events, which may occur, however infrequently, in modern data centers (see, e.g., [1, 2, 17, 27]).

To the best of our knowledge, none of the approaches in the state-of-the-art satisfy all of the above requirements at the same time.

Table 1: Approaches to construct pub/sub TCO

		Knowledge	Churn Handling	Runtime	Avg Degree	Max Degree
OMen		Local/Global	✓	Fast	$\approx O(\rho \log \lvert V\rvert T\rvert)$	$\approx O\left((\lvert V\rvert/\rho) \log \lvert V\rvert T\rvert\right)$
Centralized Algorithms	LowODA [21]	Global	✗	Slow	$O(\rho \log \lvert V\rvert T\rvert)$	$O\left((\lvert V\rvert/\rho) \log \lvert V\rvert T\rvert\right)$
	GM [6]	Global	✗	Slow	$O(\log \lvert V\rvert T\rvert)$	$\Theta(\lvert V\rvert)$
	MinMaxODA [21]	Global	✗	Slow	$\Theta(\lvert V\rvert)$	$O(\log \lvert V\rvert T\rvert)$
Decentralized Protocols	[7, 11, 22, 29]	Local/Global	✓	Fast	Unknown	Unknown

Table 1 classifies existing approaches into two categories: (i) the centralized algorithms that statically construct provably low-degree TCOs from scratch and (ii) the decentralized protocols that strive to dynamically maintain low node degrees (and in many cases TCOs) in a best-effort fashion. Unfortunately, the former are runtime-costly and problematic for fast recovery [R2], decentralized maintenance [R3] and load balancing [R4], which makes them unsuitable as a dynamic solution. On the other hand, the latter produce significantly higher node degrees than the former, as shown in [7, 29], thereby violating [R1].

In contrast, we propose OMen, a hybrid approach, which lies between the centralized algorithms and the decentralized protocols, combining the strengths of both. A principle challenge for our design is to handle the departure of nodes: Even a single node failure may leave the overlay topic-disconnected for each of the potentially many topics the departed node was interested in. In such a scenario, centralized schemes try to find a globally optimal way to restore the TCO with minimum increase in node degrees, by taking the interests of all nodes and their correlation into account and performing a relatively expensive computation. In contrast, decentralized schemes repair the overlay locally and independently for each neighbor of the failed node, without synchronization with other nodes. This can be done fast, but may lead to a significant rise in node degrees.

We propose a middle ground, where a limited number of nodes coordinate under churn in the decentralized environment: OMen proactively maintains a backup set for each node, which is bigger than the set of neighbors, but much smaller than the set of all the nodes. If a node fails, topic-connectivity is reactively restored using its backup set. The protocol for overlay mending employs weak coordination, which is fast and fully restores TCO in face of up to a configurable number of concurrent churn events, at the risk of adding redundant overlay links upon concurrent churn events. An additional novelty of this work is the decentralized scheme for maintaining the backup set. In particular, OMen requires the backup set to guarantee *topic coverage*, i.e., the backup set for node v' should cover all subscribed topics of v'. This prerequisite imposes unavoidable bias on the selection of backup nodes and negatively impacts the balance of load for backup sets across all nodes. Uniformly random assignment is no longer enough to reconcile building backup sets; if the distribution of node interests is skewed, it might be impossible to avoid skewness in the distribution of backups. We strive to keep the backup size sufficiently large for overlay quality yet small enough for efficient computation, take into account heterogeneous interests of individual nodes, achieve load balancing across the backups despite this heterogeneity, and do the above in presence of concurrent churn. Since maintenance of the backup set is proactive and running in the background, it speeds up the repair time [R2] compared to state-of-the-art protocols, as we show experimentally.

We hide the complexity of the churn handling mechanism by exporting simple and straightforward APIs to support pub/sub routing. We present OMen with complete architecture and protocol design in §3, §4, and §5. We conduct comprehensive experiments in §6 under a variety of pub/sub workloads. We use both synthetic workloads and large-scale real-world ones (i.e., Facebook [32] and Twitter [16]) with up to 10K nodes, 10K topics and 5K subscriptions per-node. We generate churn traces from a month-long Google cluster data trace [1]. Our results demonstrate good performance of OMen with regards to the requirements above.

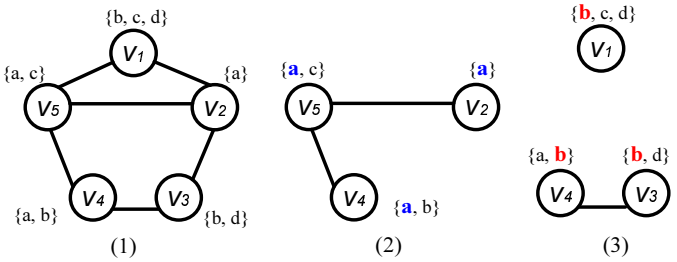


Figure 1: (1) An overlay G. (2) Subgraph $G^{(a)}$ is topic-connected. (3) Subgraph $G^{(b)}$ is not topic-connected.

2 Preliminaries

We abstract a distributed topic-based pub/sub system as an instance (V, T, I), where V is the set of nodes, T is the set of topics, and I is the interest function such that $I : V \times T \rightarrow \{true, false\}$. Node $v \in V$ is interested in topic $t \in T$ iff $I(v, t) = true$. We also say that node v subscribes to topic t.

We regard the pub/sub overlay for (V, T, I) as an undirected graph $G = (V, E)$ over the node set V with the edge set $E \subseteq V \times V$. The sub-overlay *induced* by $t \in T$ is a subgraph $G^{(t)} = (V^{(t)}, E^{(t)})$ such that $V^{(t)} = \{v \in V | I(v, t)\}$ and $E^{(t)} = \{(v, w) \in E | v \in V^{(t)} \wedge w \in V^{(t)}\}$. If $G^{(t)}$ contains at most one connected component for each topic $t \in T$, then $G = (V, E)$ forms a *topic-connected overlay* (TCO) for (V, T, I), which we denote as $TCO(V, T, I, E)$. Please see Fig. 1.

3 Overview of OMen

This section gives the intuition of OMen, our design decisions, rationales and novelties, before considering implementation details. §4 presents a simplified version of OMen that only handles simple churn events. §5 describes enhancements to the base protocol needed to handle concurrent churn events.

3.1 Scope of OMen

OMen incorporates churn handling mechanism and allows any node to join or leave (gracefully or by crashing) at any time. We focus on node crashes, whose handling poses unique challenges for the five requirements listed in §1. Like many practical systems [9, 17, 27], OMen prioritizes handling of node churn (join, leave or fail), while assuming reliable communication within the data center.

The target environment is a large-scale data center with moderate churn [1, 2, 17, 27]:

1. Most churn events are *simple*, i.e., only one node joins, leaves, or fails at one time. Concurrent churn events involving multiple

nodes occur infrequently.

2. Intervals between successive churn events are in the order of tens of minutes, depending on the cluster size.

Therefore, simple churn events are the most common cases, which we should handle efficiently. Concurrent churn events are possible yet infrequent. They must be handled correctly (i.e., topic-connectivity must be restored); however, we do not need to optimize performance for them.

3.2 Main concepts and mechanisms

We exploit the centralized TCO design algorithms [6, 21], because they guarantee low node degrees in the output TCO [R1]. However, it is challenging to apply these algorithms in dynamic settings. In particular, these static algorithms exhibit prohibitively high running time complexity and require global knowledge, which are detrimental to [R2] and [R3], respectively. On the other hand, purely decentralized protocols [7, 22, 11, 29] build overlays of a lower quality. To provide a useful middle ground, OMen only employs a carefully selected subset of nodes, called *shadow set*, to repair the overlay under churn.

To realize this shadow-based strategy, we should answer two central questions: (I) *who* executes the shadow-based algorithm to repair the overlay for each simple churn event and (II) *how* to select the shadow set, which is the key for treading the balance between good overlay quality [R1] and efficient running time [R2].

With regard to Question (I), we want to evenly amortize the burden of churn handling across all nodes to meet the requirements of decentralization [R3] and fairness [R4]. OMen selects a churn coordinator for each simple churn event. The main responsibilities of each churn coordinator consist of: (1) selecting the shadow set upon each simple churn event (see Question (II) above and our discussion later in this section), (2) collecting the information from the shadow set, (3) executing the centralized overlay repair algorithm, (4) propagating the knowledge about new edges to the nodes that need to establish these edges, and (5) orchestrating the repair within a small set of other nodes in case of concurrent churn events (see §3.3). The coordinator *selection* mechanism that we adopt is commonly used in many fault-tolerant applications based on DHTs [10, 15, 28]. As a result, each OMen node only serves as the coordinator for a limited number of churn events and needs to maintain a small amount of information [R3, R4].

With regard to Question (II), the churn coordinator must carefully and efficiently select the shadow set, especially under node leaves. The departure of a node may render the overlay disconnected for many topics, and it is difficult to gather information necessary for churn handling on the fly upon node leaving. However, in order to run the overlay repair algorithm, the coordinator demands a certain amount of information, including disconnected topics, a shadow set, and subscriptions of all shadow nodes. On the other hand, the churn coordinator does not possess global knowledge, and all information is distributed over the network - a requirement of decentralization [R3]. Thus, the coordinator needs to collect this information from the shadow set, which is, however, time consuming. To accelerate responsiveness under churn [R2], OMen initiates these information collection procedures before the occurrences of churn events. In our design, a coordinator responsible for a node v' proactively pre-computes the shadow set for v' and gathers information from the set in advance, in preparation for the possible future departure of v'. For each node $v' \in V$, the coordinator maintains (1) the TCO neighbors of v', $N(v')$, and (2) a *backup set* for v', $B(v')$. When node v' leaves or fails, we simply compute the shadow set S for this churn event by combining the backup set and the TCO neighbors of v', i.e., $S = B(v') \cup N(v')$. Similarly, when node v' joins, the shadow set is computed as $S = B(v') \cup \{v'\}$.

Since the backup set is an essential component of shadows set, the problem of building backup sets plays a vital role in the context of Question (II). First, just like the shadow set, the backup set must capture the trade-offs between overlay quality [R1] and recovery time [R2]. Second, the backup set should be of moderate size to ensure scalability of the churn handling protocol, as demanded by decentralized maintenance [R3]. Third, fairness [R4] requires nodes to evenly contribute to the backup sets, which may conflict with the inherent skewness in the pub/sub workloads.

To address all these concerns, we identify two important properties for backup sets, namely *topic coverage* and *individual commitment* (see definitions in §4.2). These properties have guided our algorithm development. Even computing one optimal backup set is NP-hard (see §4.2). Fortunately, we can obtain approximated ones efficiently only with partial knowledge [R3, R4], and these backup sets turn out to perform well with respect to both overlay quality [R1] and time efficiency [R2].

We employ *gossiping* to build backup sets for all nodes in a distributed manner [R3]. First, since backup sets do not need to be optimal and can be selected in many different ways, this flexibility allows for both independent construction of backup sets in parallel for different nodes and tolerance to the element of randomness inherent to gossiping. Second, the random nature of epidemic protocols helps alleviate the bias that stems from inherent skewness in the pub/sub workloads, thus, balancing the overall distribution of backup sets [R4]. Third, gossiping is highly resilient to churn and capable of quickly refreshing backup sets under churn events [R2]. Forth, our proactive construction of backup sets ahead of churn events makes OMen tolerant to convergence time of asynchronous gossiping protocol [R2].

3.3 Handling concurrent churn events

In practice, OMen needs to deal with concurrent churn events [R5]. OMen exports a configurable system parameter $\mathbb{L}$, the *resilience factor*. Protocols operating atop OMen (e.g., pub/sub routing) can specify $\mathbb{L}$, and OMen adjusts its components accordingly to guarantee the TCO property under up to $\mathbb{L}$ concurrent churn events.

We regard concurrent churn events as a set of simple churn events and handle each of them independently. Recall that for each simple churn event, only one node joins, leaves or fails. Informally speaking, two simple churn events are concurrent, if the second one occurs before the handling of the first one is completed.

For each simple churn event, OMen guarantees that at least one coordinator will eventually complete the churn handling procedure. Like many DHTs [10, 15, 28], OMen forms an identifier circle, and each node has immediate predecessor and successor. Suppose we have a network of N nodes, i.e., $V = \{v_1, v_2, \cdots, v_N\}$, where v_{i+1} is the immediate successor of v_i, $1 \leq i < N$, and v_1 is the immediate successor of v_N. A simple churn event occurs at node v_1; by default, the coordinator is node v_2, the closest successor of v_1 that stays alive. If v_2 leaves before it completes handling the churn event at v_1, then v_3 takes over and becomes the churn coordinator for the concurrent churn events at v_1 and v_2. In general, if a set of concurrent churn events occur at $\{v_1, v_2, \cdots, v_l\}$, then node v_{l+1} will serve as the coordinator for all of them, where $l \leq \mathbb{L}$ by assumption.

To fulfill the above design, OMen performs the following three tasks for handling concurrent churn events. First, each node prepares the required knowledge (i.e., backup sets and TCO neighbors) for its $\mathbb{L}$ immediate predecessors. Second, we develop an automatic failover mechanism for the churn coordinator. After detecting a churn event *chev* at v', OMen propagates the notification about *chev* to $\mathbb{L}$ successors of v', instead of just one. All these suc-

cessors keep a record for *chev*, in case that the default coordinator fails before handling *chev*. Third, OMen has a garbage collection process for cleaning the churn events that have been handled and outdated knowledge in the local view.

We adopt a relaxed consistency for the coordinator synchronisation: it is possible but unusual that multiple (i.e., more than one) nodes regard itself as the coordinator and thus handle the same simple churn event. The only penalty is that a few redundant edges can be introduced. We believe it is better to deal with this insignificant redundancy for the sake of faster overlay repair [R2].

3.4 Novelties

Our OMen design exhibits novelties in several respects:

1. OMen presents a reasonable middle ground between centralized algorithms and decentralized protocols for pub/sub overlay maintenance under churn. OMen achieves both overlay quality [R1] and time efficiency [R2], whereas previous work only optimizes one metric but performs poorly with regard to the other.

2. We are among the first to introduce the idea of a backup set for pub/sub overlay design. While backup nodes have been extensively used in fault-tolerant solutions, few employ them for peer-to-peer overlay recovery – none is designed for pub/sub. As compared to other distributed applications, pub/sub exhibits more complexity and unique requirements for the backup set construction, e.g., the properties of topic coverage and individual commitment. Particularly, our algorithms for building backup sets satisfy these properties while performing well at the entire system level in terms of overlay quality [R1], efficiency [R2], decentralization [R3], and fairness [R4].

3. We proactively build backup sets using a gossip-based mechanism. Although a few distributed systems such as [19] adopt epidemic protocols to maintain backup nodes, the backup sets maintained in [19] are independent, and thus a naive random selection algorithm is good enough. In OMen, however, it is challenging to integrate the algorithms of building a backup set with a gossip mechanism: the backup set selection at one OMen node impacts the backup set selection at other nodes, and we have to leverage the randomness of gossiping to balance the selection bias that comes from inherent skewness of pub/sub workloads.

4. Thanks to the semantics of pub/sub overlay repair, we adopt a loose consistency model for concurrent churn handling. As a result, we do not have to resort to strict consistency protocols (e.g., Paxos [18] and Raft [24]), which are unsatisfactory in terms of time efficiency [R2] due to considerable communication overhead.

Please see a more detailed discussion in §7 to better distinguish our work from a broad range of related approaches.

4 The Base OMen Protocol

4.1 Shadow-based overlay repair

The concept of *shadow set* lies at the core of our mechanism for churn handling. Upon each simple churn event, we select a *shadow set*, i.e., a "proper" subset of all nodes, for repairing the TCO. Basically, we execute the centralized algorithms on the shadow set (instead of the entire node set) while respecting the existing overlay edges. We adopt centralized TCO design algorithms [5, 6, 21], because they guarantee low node degrees [R1] while purely decentralized protocols [7, 22, 11, 29] do not. However, it is difficult to employ these algorithms in dynamic settings; in particular, these static algorithms suffer from expensive running time complexity and require complete global knowledge, which are against efficiency [R2] and decentralization [R3], respectively. Our objective is to avoid imposing these costs onto the dynamic infrastructure. In principle, we can alleviate these shortcomings by feeding only a small subset of nodes and their subscriptions to the computation and reusing existing edges. We show analytically and empirically that it is promising to keep the node degrees low [R1] while significantly improving the running time efficiency [R2] – the shadow set serves as a tuning knob to strike a balance between these two conflicting objectives.

Alg. 1 Shadow-based overlay repair algorithm

```
repairTcoOnChurn(S, T_chev, I|_S, E_cur)
1: E_new ← ∅, E_pot ← (S × S)\E_cur
2: while G = (S, E_new ∪ E_cur) is not TCO for (S, T_chev, I|_S) do
3:     e ← select an edge according to LowODA
4:     E_new ← E_new ∪ {e}, E_pot ← E_pot − {e}
5: return E_new
```

Alg. 1 presents our implementation for the shadow-based strategy. It iteratively adds carefully selected edges one by one until attaining a TCO. At each iteration, Line 3 selects an overlay edge based on the LowODA-rule [21]. Thanks to the properties of LowODA, we derive that Alg. 1 can efficiently repair the TCO and achieve node degrees close to those of LowODA upon a simple churn event.

Given (V, T, I), we denote by $D_{\min}(V, T, I)$ (and $\bar{d}_{\min}(V, T, I)$) the minimum possible maximum (and average) node degree of any TCO for (V, T, I). We denote by $D(V, E)$ and $\bar{d}(V, E)$, the maximum and average degree of a graph $G = (V, E)$, respectively.

$$D_{\min}(V, T, I) = \min_{E:TCO(V,T,I,E)} D(V, E),$$
$$\bar{d}_{\min}(V, T, I) = \min_{E:TCO(V,T,I,E)} \bar{d}(V, E).$$

Lemma 1. *Given ρ as a parameter for LowODA, Alg. 1 has the following properties under one simple churn event:*
(a) the running time is $O(|S|^2.|T_{chn}|)$,
(b) the output overlay is topic-connected,
(c) the maximum node degree $D(V', E_{cur} \cup E_{new})$ is $O\left(D(V, E_{cur}) + D_{\min}(S, T_{chn}, I|_S) \cdot \frac{|S|}{\rho} \log(|S||T_{chn}|)\right)$,
(d) the average node degree $\bar{d}(V', E_{cur} \cup E_{new})$ is $O\left(\bar{d}(V, E_{cur}) + \bar{d}_{\min}(S, T_{chn}, I|_S) \cdot \rho \log(|S||T_{chn}|)\right)$.

Proof sketch: We devise an indexing data structure that improves LowODA's running time from $O(|V|^4|T|)$ to $O(|V|^2|T|)$ [5]. The approximation ratios stem from Theorem 1 and 2 of [21], and the proofs are similar to the one we provided for Lemma 1 in [5]. ■

Lemma 1 quantitatively describes how the shadow set governs the trade-offs between the output quality and time efficiency of Alg. 1, which impacts [R1] and [R2] of OMen.

One of the main challenges that OMen addresses is selection of an appropriate shadow set. The size of the shadow set plays a chief role in establishing overlay quality [R1], recovery speed [R2], load balance across the nodes [R4], etc. Consider the possible shadow sets that are capable of restoring TCO when node v' is leaving. On the one hand, it is always safe to pick the entire node population $V = V\backslash\{v'\}$ as the shadow set. However, a large shadow set exhibits two serious drawbacks: (1) expensive running time [R2], since the complexities of many TCO design algorithms [5, 6, 21] are quadratic in the number of nodes; and (2) poor decentralization [R3], because a larger set requires more knowledge and coordination of nodes. On the other hand, it is sufficient to add edges among the TCO neighbor set of v', $N(v')$, to the existing overlay. This can be done promptly, since $N(v')$ is usually of much smaller cardinality than the complete node set V'; however, the node degrees of $N(v')$ would degrade significantly in the output TCO [R1].

4.2 Backup set – a subset of shadows

To realize the shadow-based strategy, we need a careful design for shadow set selection, especially under node leaves. As we have briefly discussed in §3.2, to prepare for the potential departure of some node v', a preassigned coordinator proactively builds the backup set for v' and collects associated information in advance.

For each v', we define the backup set $B(v')$ to be *acceptable* if $B(v')$ satisfies the following two desirable properties:

(i) *Topic coverage*: $B(v')$ covers all subscribed topics of v',

$$v'.topics \subseteq \bigcup_{u \in B(v')} u.topics \tag{1}$$

(ii) *Individual commitment*: each backup node $u \in B(v')$ shares at least one topic with v',

$$\forall u \in B(v'), (u.topics \cap v'.topics) \neq \emptyset \tag{2}$$

The topic coverage property ensures that the backup set $B(v')$ (and hence the shadow set) contains sufficient nodes to restore the TCO under node leaves or failures. The individual commitment property helps keep the backup set small by eliminating unnecessary redundancy – nodes share no common interest with v' have zero contribution towards TCO repair under the departure of v'.

Still, there is a trade-off concerning the size of the backup set. On the one hand, we prefer small cardinality in the backup set for efficiency [R2], because the number of backups directly impacts the size of the shadow set and thus the time complexity of the overlay repair algorithms. On the other hand, a sufficiently large backup set helps to ensure the quality of the output TCO [R1]: a larger backup set means a large shadow set S and therefore a higher probability for the instance $(S, v'.topics, I|_S)$ to approximate (V', T, I') with regard to node degrees.

We introduce the *coverage factor* to tune the size of the backup set for a node and seek balance between the time complexity of the overlay maintenance [R2] and the quality of the output TCO [R1]. Given (V, T, I), we build the backup set $B(v')$ for each node $v' \in V$. The coverage factor for the backup set $B(v')$, denoted as $\lambda(B(v'))$ (or λ), is the minimum number of subscribers to t within $B(v')$ taken across all $t \in v'.topics$:

$$\lambda(B(v')) = \min_{t \in v'.topics} \left| \{u | u \in B(v') \wedge t \in u.topics\} \right| \tag{3}$$

The coverage factor is an integer ($\lambda \geq 0$) s.t. the backup set $B(v')$ covers each topic interest of v' at least λ times.

The coverage factor selection exhibits the trade-off between the running time [R2] and node degrees [R1]: On the one hand, $\lambda = 0$ minimizes the size of the backup set and running time, but leads to a severe impact on the node degrees. On the other hand, if we choose the coverage factor to be a large value such that all nodes of V' have to be included in the backup set, then, both the maximum and average node degrees are close to those generated by running the static algorithms from scratch, but the runtime cost is not insignificant. Our experiments in §6.4 show that: (1) an increase in λ beyond 5 only marginally improves the node degrees under churn; and (2) the backup set for $\lambda = 5$ is significantly smaller than the complete node set itself. We therefore choose 5 as the default value for λ.

Besides, the coverage factor, λ adds some redundancy to the backup set so that OMen is sustainable under concurrent churn events. With the coverage factor λ, the backup set $B(v')$ remains acceptable as long as less than λ nodes in $B(v')$ fail.

An acceptable backup set $B(v')$ is equivalent to a *set cover* where v''s topics are the universe of ground elements that are supposed to be covered by the topic set of backup nodes. The *minimum set cover* problem is a well-studied NP-hard problem [8, 12]. We can apply classical algorithms to obtain an acceptable (not necessarily optimal) backup set.

There exist two efficient approximation algorithms for the *minimum set cover* problem: the *greedy* algorithm [8] and the *primal-dual* algorithm [12]. Accordingly, we have two implementations for `buildBackups`() - each executes the greedy or primal-dual algorithm iteratively until obtaining a *λ-backup-set*, which we refer to as `buildBackupsGreedily`() and `buildBackupsByPD`(), respectively.

The `buildBackupsGreedily`() procedure applies the *greedy* set cover heuristic for constructing a backup set [8]. It always chooses the node w that covers the maximum number of uncovered topics. This algorithm achieves a logarithmic approximation ratio. However, its greediness leads to prioritizing bulk nodes that subscribe to a large number of topics upon backup selection. A small number of bulk nodes would serve as backups for a large number of nodes while the majority of lightweight nodes would not be selected as backups at all. Fairness [R4] is lost to a large extent in this case. Furthermore, the impact of accumulated sub-optimality would become progressively severe over time as more churn events occur, which is deficient for overlay quality [R1]. We can mitigate these disadvantages by using randomization in the algorithm, which leads us to `buildBackupsByPD`().

The `buildBackupsByPD`() procedure uses the *primal-dual* method for computing the set cover [12]. The algorithm proceeds in an iterative manner: each time randomly picking an uncovered topic t and choosing a subscriber w for t with the maximum uncovered topics as a backup. The primal-dual algorithm yields an f-approximate solution for minimum set cover where f is the maximum frequency of any element. The approximation ratio of primal-dual is higher than that of the greedy set cover algorithm. Yet, it is deemed acceptable in practice for many instances of the problem. Moreover, the primal-dual approach integrates randomness into greediness and effectively mitigates the prioritization of bulk subscribers. Therefore, we decide to leverage the primal-dual algorithm towards building the backup set. We experimentally compare these two implementations and verify our choice in §6.3.

4.3 Preparing backup sets by gossiping

OMen maintains at each node a local view, which prepares backup sets and other information for all nodes in case of churn. The churn coordinator scheme allows us to distribute all knowledge across the network. To cope with simple churn events, each node v only needs to store information for its immediate predecessor v', including three local view variables in Alg. 2: (1) the backup set for v' in $v.B(v')$, (2) the TCO neighbors of v' in $v.N(v')$, and (3) the ready flag for v' in $v.R(v')$.

Alg. 2 Local view at node $\mathbf{v} \in V$

▷ $\mathbf{v}.B$: a hashtable that maps node v' (as a key) to the backup set for v' (as a value): $\mathbf{v}.B(v')$ is the backup set that $\mathbf{v}$ maintains for v'.
▷ $\mathbf{v}.N$: a hashtable that maps node v' (as a key) to the TCO neighbor set of v' (as a value): $\mathbf{v}.N(v')$ is the TCO neighbors that $\mathbf{v}$ maintains for v'.
▷ $\mathbf{v}.R$: a hashtable that maps node v' (as a key) to the ready flag for v' (as a value): $\mathbf{v}.R(v')$ is the ready flag that $\mathbf{v}$ maintains for v'.

We implement this local view layer by *gossiping* [13, 14]. First, backup sets have certain degrees of flexibility and fuziness, as Eq. (1), (2), and (3) jointly define. Second, gossiping is fast [R2], decentralized [R3], load balanced [R4], and reliable [R5].

We use T-Man [14], a generic gossiping framework for constructing and maintaining a wide range of topologies. The T-Man instance consists of two threads at each node (see Alg. 3): (1) an

Alg. 3 T-Man framework at $\mathbf{v} \in \mathbf{V}$

// active thread

1: **upon** a random time once in each consecutive interval of T_{wait}
2: $p \leftarrow$ select a peer from local view
3: $buffer \leftarrow \mathbf{v} \cup \mathbf{v}.\texttt{getView}() \cup rnd.view$
4: send $buffer$ to node p
5: receive $buffer_p$ form node p
6: $\texttt{selectView}(\mathbf{v}.\texttt{getView}() \cup buffer_p)$

// passive thread

1: **upon** receive $buffer_q$ from q
2: $buffer \leftarrow \mathbf{v} \cup \mathbf{v}.\texttt{getView}() \cup rnd.view$
3: send $buffer$ to node q
4: $\texttt{selectView}(\mathbf{v}.\texttt{getView}() \cup buffer_q)$

Alg. 4 T-Man functions at node $\mathbf{v} \in \mathbf{V}$

▸ $\mathbf{v}.\texttt{getView}()$

1: **return** all local view from both $\mathbf{v}.B$ and $\mathbf{v}.N$

▸ $\mathbf{v}.\texttt{selectView}(candiates)$

1: **for all** predecessor v' **do**
2: $\mathbf{v}.N(v') \leftarrow$ get the TCO neighbor set of v'
3: $\langle v.B(v'), \mathbf{v}.R(v') \rangle \leftarrow \texttt{buildBackups}(v', candiates)$ // see §4.2

active thread periodically initiating communication with other nodes and (2) a passive one responding to incoming gossip messages.

Each node v periodically exchanges a buffer with another node p, randomly chosen from the current local view. Node v then updates its local view using a combination of its current view and $buffer_p$ from peer p. The same process takes place at other nodes (e.g. p).

As Line 3 in the active thread and Line 2 in the passive thread show, each node v obtains a gossip buffer by merging (1) v's self descriptor, (2) v's local view, and (3) $rnd.view$, a random sample of the nodes from the entire network, which is provided by a *peer sampling service* [14]. This service is to provide every node with peers to exchange information with. These peers should perform like being selected uniformly at random from the set of all nodes.

Using T-Man, each node v periodically exchanges information with another node and updates the local view. Alg. 4 instantiates two T-Man methods: `getView()` and `selectView()`. Function `getView()` computes the current local view, which is the union of backup sets and neighbor sets. Function `selectView()` decides which nodes to keep for each gossip exchange. Given an input candidate set, $v.\texttt{selectView}()$ chooses nodes that are suitable for filling up $v.B$ and $v.N$ (see the algorithm design in §4.2). In T-Man, each node v obtains such a candidate set by merging (1) the local view of v and (2) the buffer received from another node p.

Since refining backup sets is a periodic process performed in the background, it does not automatically guarantee that the backup set will be ready when there is a need to compute the shadow set for overlay repair. If it is not ready, T-Man will continue the exchange that improves the quality of the local view and ensures fast convergence and high robustness in dynamic environments. While the repair process is blocked during this period, the pause usually does not last long because (a) the exchange starts proactively ahead of the failure and (b) after a specific timeout multiple successors will attempt to handle the event in the more advanced OMen (see §5).

It is worth noting that the local view layer of OMen exhibits several distinguishing characteristics as compared to typical T-Man applications (see 7.3).

Alg. 5 Churn handling at node $\mathbf{v} \in \mathbf{V}$

▸ $\mathbf{v}.\texttt{handleChurnEvent}(chev)$ // $v' = chev.node$

1: **if** $\mathbf{v}.\texttt{isCoordinator}(v')$ **then**
2: **if** $chev.type$ is *Leave* **then**
3: $unassigned \leftarrow$ remove from $\mathbf{v}.standbyL$ all unfinished events that assigned v' as the coordinator
4: add $unassigned$ to $\mathbf{v}.exeQ$
5: add $chev$ to $\mathbf{v}.exeQ$
6: **else** // $\mathbf{v}$ is not the coordinator for $chev$ at v'
7: add $chev$ to $\mathbf{v}.standbyL$, start timer for $chev$ // cf. `processStandbyL()`

// process churn events in $\mathbf{v}.exeQ$ one by one sequentially w.r.t. local time

▸ $\mathbf{v}.\texttt{processExeQ}()$

1: **upon** $\mathbf{v}.exeQ \neq \emptyset$
2: $chev \leftarrow$ remove from $\mathbf{v}.exeQ$ the event with the smallest $localtime$
3: **wait until** $\mathbf{v}.R(v') \vee \neg\mathbf{v}.\texttt{isInPredRange}(v')$ // $v' = chev.node$
4: **if** $\mathbf{v}.R(v') \wedge \mathbf{v}.\texttt{isInPredRange}(v')$ **then**
5: $\mathbf{v}.\texttt{repairTcoOnChurn}(chev)$
6: $\mathbf{v}.\texttt{notifySuccRange}(chev)$
7: **else** // v' is out of the $\mathbb{L}$-predecessor range of $\mathbf{v}$
8: remove $chev$ from $\mathbf{v}.exeQ$, add $chev$ to $\mathbf{v}.standbyL$

// process timeout standby churn events in $\mathbf{v}.standbyL$

▸ $\mathbf{v}.\texttt{processStandbyL}()$

1: **upon** Timeout for $chev \in \mathbf{v}.standbyL$
// cf. Line 7, `handleChurnEvent()`
2: **if** $\mathbf{v}.\texttt{isInPredRange}(v')$ **then** // $v' = chev.node$
3: add $chev$ to $\mathbf{v}.exeQ$, remove $chev$ from $\mathbf{v}.standbyL$

// clean churn events that are handled in $standbyL$

▸ $\mathbf{v}.\texttt{gcChurnEvent}(chev)$

1: remove $chev$ from $\mathbf{v}.standbyL$
2: clean local view $\mathbf{v}.B$, $\mathbf{v}.N$, and $\mathbf{v}.R$

4.4 Example

We exemplify how OMen handles a simple churn event. In Table 2 and Fig. 2, we have an instance of 12 nodes with 6 topics where $V = \{0, 1, \ldots, 11\}$ and $T = \{a, b, c, d, e, f\}$. Fig. 2 labels the interests for all nodes, which collectively define the interest function I.

In Fig. 2(a), an initial TCO is constructed and the local view is built for each node. Table 2(a) shows that, node v_{11} maintains some information for v_{10}, including (1) TCO neighbors placed in $v_{11}.N(v_{10})$ and (2) the backup set built in $v_{11}.B(v_{10})$.

In Fig. 2(b), node v_{10} departs from the overlay, and node v_{11} is the immediate successor of the churn node v_{10} and thus the churn coordinator for this churn event. Thanks to the local view prepared ahead of time, OMen can readily compute the shadow set at coordinator v_{11}, i.e., $S = v_{11}.N(v_{10}) \cup v_{11}.B(v_{10}) = \{v_1, v_3, v_5, v_6, v_{11}\}$. Coordinator v_{11} executes the shadow-based algorithm locally and repairs the overlay (see algorithm design and analysis in §4.1).

Fig. 2(c) illustrates the results after OMen properly handled the departure of v_{10}. First, OMen adds three new overlay edges $E_{new} = \{(v_{11}, v_1), (v_{11}, v_5), (v_5, v_6)\} \subseteq S \times S$, re-attaining the TCO. Second, OMen stablizes the local view at each node accordingly. In Table 2(c), v_{11} updates its immediate predecessor as v_9 and prepares $v_{11}.N(v_9)$ and $v_{11}.B(v_9)$ for the potential departure of v_9.

In Table 2(a), $v_{11}.B(v_{10})$ is a backup set for v_{10} with $\lambda = 1$, which covers c and f only once. In Table 2(c), $v_{11}.B(v_9)$ covers each $t \in v_9.topics$ at least twice – a backup set for v_9 with $\lambda = 2$.

5 Handling concurrent churn events

Upon each simple churn event, OMen is able to stabilize through an entire churn handling procedure, which consists of the following: (i) churn handling, (ii) local view stabilization, and (iii) iden-

Table 2: The *states* of the churn coordinator v_{11} in the running example

(a) before v_{10}'s departure

Local fields of $\mathbf{v_{11}}$

neighb	$\{v_0, v_{10}\}$
exeQ	{}
$B(v_9), B(v_{10})$	NIL, $\{v_1, v_6\}$
$N(v_9), N(v_{10})$	NIL, $\{v_3, v_5, v_{11}\}$
$R(v_9), R(v_{10})$	NIL, *true*
getPreds()	$\{v_{10}\}$

(b) handling v_{10}'s departure

Local fields of $\mathbf{v_{11}}$

neighb	$\{v_0, v_{10}\}$
exeQ	$\{(v_{10}, \text{Leave})\}$
$B(v_9), B(v_{10})$	NIL, $\{v_1, v_6\}$
$N(v_9), N(v_{10})$	NIL, $\{v_3, v_5, v_{11}\}$
$R(v_9), R(v_{10})$	NIL, *true*
getPreds()	$\{v_9\}$

(c) v_{10}'s departure handled

Local fields of $\mathbf{v_{11}}$

neighb	$\{v_0, v_1, v_5\}$
exeQ	{}
$B(v_9), B(v_{10})$	$\{v_1, v_4, v_5, v_7\}$, NIL
$N(v_9), N(v_{10})$	$\{v_1, v_4, v_7\}$, NIL
$R(v_9), R(v_{10})$	*true*, NIL
getPreds()	$\{v_9\}$

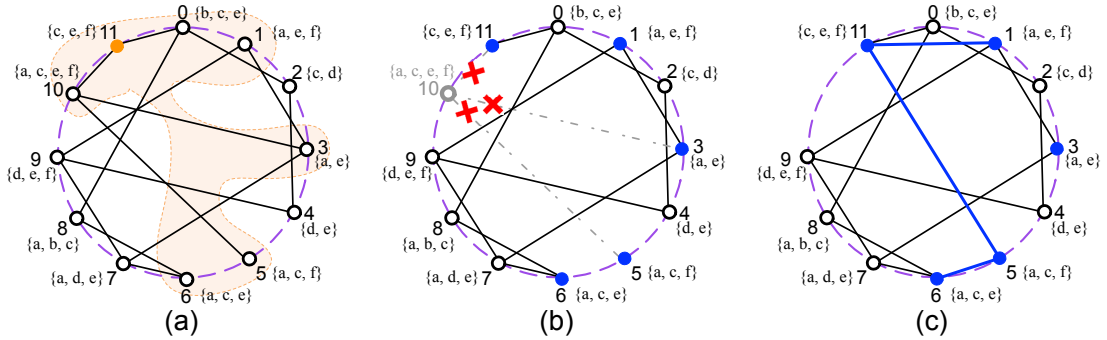


Figure 2: Example: OMen handles a simple churn event - the departure of node v_{10}.

Alg. 6 Auxiliary functions at node $\mathbf{v} \in V$

public:

▸ $\mathbf{v}$.getPreds(): get the the list of $\mathbb{L}$ predecessors

▸ $\mathbf{v}$.isInPredRange(v'): check whether v' is in $\mathbf{v}$'s $\mathbb{L}$-predecessor range

▸ $\mathbf{v}$.isCoordinator(v'): check whether $\mathbf{v}$ is the coordinator for the churn event of node v', i.e., the immediate successor of v'

▸ $\mathbf{v}$.notifySuccRange(*chev*): notify the $\mathbb{L}$-successor range of the churn node that the coordinator completes overlay repair for *chev*

tifier circle adjustment.

Each simple churn event *chev* has a life cycle, which starts when *chev* occurs and ends when the entire churn handling procedure for *chev* finishes. We denote by $\tau_s(chev)$ and $\tau_e(chev)$ the start and end time for the life cycle of *chev*, respectively. Two simple churn events are called concurrent, if their life cycles overlap with each other. Suppose a simple churn event *chev* occurs at $\tau_s(chev)$, we define the *concurrent churn event set* w.r.t. *chev*, denoted by $\mathbb{CCE}(chev)$, to be the set of all simple churn events that are concurrent to *chev*.

As previously mentioned, we assume that the size of the concurrent churn event set does not exceed $\leq \mathbb{L}$. Formally,

ASSUMPTION 0. $|\mathbb{CCE}(chev)| \leq \mathbb{L}, \forall chev.$

The main property of OMen is that it always effectively *stabilizes* under Assumption 0.

Alg. 5 specifies the basic churn handler, which is invoked upon each churn event. Alg. 6 lists all auxiliary functions for churn handling. For each simple churn event *chev* at node v', OMen will invoke function v.handleChurnEvent(*chev*) on at least one node v, which was in the $\mathbb{L}$-successor range of v' when *chev* occurred and remains in the range throughout the entire churn handling procedure. Typically, node v is the immediate successor of v', unless this immediate successor fails or other nodes join in between v' and its immediate successor. Further, handleChurnEvent(*chev*) will only be invoked on nodes that were in the $\mathbb{L}$-successor range at some point during the life cycle of *chev*.

The handler maintains $v.exeQ$ and $v.standbyL$ for handling concurrent churn. If a node regards itself as the coordinator for *chev*, i.e., the immediate successor for *chev.node*, it places *chev* in $exeQ$. Otherwise, *chev* is placed into $standbyL$. If a subsequent failure of the coordinator is detected and a node considers itself as a new coordinator for *chev*, it will remove *chev* from $standbyL$ and place it into $exeQ$. When a coordinator successfully completes overlay repair for *chev*, it invokes notifySuccRange(), which will eventually lead to garbage collection and removal of *chev* from $standbyL$ of other nodes. Note that it is possible that handleChurnEvent(*chev*) will not be invoked on the coordinator because the latter joined after *chev* occurred. In this case, a timeout for *chev* on v will trigger the relocation of *chev* from $standbyL$ to $exeQ$ so that v will eventually handle *chev* itself.

Function v.processExeQ() processes churn events in $v.exeQ$ one by one sequentially. We dequeue *chev* from $v.exeQ$ based on a local timestamp *chev.localtime*, which OMen assigns so that all churn events locally observed can be sorted in $exeQ$. This prevents starvation that may perpetually defer the handling of some churn event. As Line 3 shows, for each $chev \in v.exeQ$, v needs to wait until either (1) the local view is ready for handling *chev*, or (2) the churn node is out of v's predecessor range. In the first case, v invokes the shadow-based overlay repair algorithm and then sends notifications about the completion of overlay repair at the coordinator (Lines 5-6). In the second case, Line 8 removes *chev* from $v.exeQ$ and adds it to $v.standbyL$, because node v is no longer responsible for *chev*.

Function v.processStandbyL() handles the churn events stored in $v.standbyL$ upon timeout, since it is possible that the coordinators did not receive notifications about the churn events. Node v relocates *chev* from $v.standbyL$ into $v.exeQ$, only if the churn node v' lies in the $\mathbb{L}$-predecessor range of v (Line 2).

Function v.gcChurnEvent(*chev*), which is invoked in Line 6 of processExeQ(), removes *chev* from $v.standbyL$ and invalidates local view entries that are no longer needed. Churn events are only delivered within a limited range of successors, so each node v only needs to keep information for a limited number of predecessors: $|\{w|v.B(w) \neq \text{NIL}\}| \leq 2\mathbb{L}, \forall v \in V$.

Table 3: Algorithms and protocols that we evaluate

OMen	**Our proposed system to maintain TCO**
OMen	Maintaining a *partial* view at each node.
OMen-G	Maintaining a *global* view at each node.
LowODA [21]	**Low Degree Overlay Design Algorithm**
LowODA-Inc	Take existing edges into account and incrementally repair TCO using the LowODA-rule.
LowODA-Re	Reconstruct TCO from scratch upon each churn event, regardless of existing edges.
SpiderCast [7]	**A peer-to-peer protocol to build TCO**
SpiderCast$_{(K_g,K_r)}$	Neighbor selection combines two *local* heuristics: *greedy* and *random coverage*. Each node tries to cover its interested topics K_g or K_r times. We use SpiderCast$(3,1)$ and SpiderCast$(4,0)$, as recommended in [7].

6 Evaluation

6.1 Experiment setup

We implement OMen and other approaches (see Table 3) in PeerSim [23]. We evaluate two versions of OMen: OMen and OMen-G, depending on whether the system maintains a partial view or a complete global view at each node. While OMen-G is impractical, we use it for sensitivity analysis wrt. the view size. As explained in §4.1, we utilize LowODA [21] as an evaluation baseline. We develop two LowODA-based implementations, LowODA-Inc and LowODA-Re. While LowODA-Re is the standard LowODA algorithm that disregards existing links and builds TCO from scratch upon every churn event, LowODA-Inc is our modification of LowODA that takes existing edges as the starting point and incrementally repairs TCO using the LowODA rule. In contrast to OMen, both algorithms are applied on the complete set of nodes. We also put SpiderCast [7] in the chart, because it is highly efficient in constructing TCOs in a decentralized peer-to-peer manner and has been adopted in practice [31]. Originally in [7], SpiderCast did not explicitly specify churn handling mechanism; however, we can simply employ SpiderCast's local neighbor selection heuristics for overlay repair under churn. More specifically, upon a churn event, each node $v \in V$ would independently invoke the neighbor selection heuristics, if the churn event renders v topic-disconnected.

Our evaluation uses both synthetic pub/sub workloads and real-world traces derived from data sets of Facebook, Twitter and Google.

Pub/Sub workloads: We synthetically generate three types of topic popularity distributions: uniform, Zipfian, and exponential. We also extract the workloads from social networks, namely Twitter and Facebook.

(1) *Synthetic workloads*: We initialize the base instance (V_0, T, I_0) with $|V_0|$=2 000, $|T|$=200 and $|v.topics| \in [10, 90]$, where the subscription size of each node follows a power law. Each topic $t \in T$ is associated with probability $p(t)$, $\sum_{\in T} p(t)$=1, so that each node subscribes to t with a probability $p(t)$. The value of $p(t)$ is distributed according to either an exponential, a Zipfian (with α=2.0), or a uniform distribution, which we call Expo, Zipf, or Unif, for short. These distributions are representative of actual workloads used in industrial pub/sub systems today [6, 7]. Expo is used by stock-market monitoring engines for the study of stock popularity in the New York Stock Exchange [30]. Zipf faithfully describes the feed popularity distribution in RSS feeds [20].

(2) *Facebook dataset*: We use a public Facebook dataset [32], with over 3 million distinct user profiles and 28.3 million social relations as a second workload for our evaluations. When a Facebook user performs some activity (e.g., sharing new photos or commenting on a blog), all her friends receive a notification as subscribed. As such, we model each user, say Alice, as a topic, and all her friends are the respective subscribers. Likewise, the friend set of Alice forms her subscription set. The Facebook relations are bidirectional, so friends in the Facebook social graph subscribe to each other in our model.

(3) *Twitter dataset*: We also use a public Twitter dataset [16], containing 41.7 million distinct user profiles and 1.47 billion social followee/follower relations. Similarly to Facebook, we model users as topics as well as subscribers. However, relations are unidirectional in Twitter, i.e., Alice following Bob does not require that Bob also follows Alice.

We extract the workloads from the original Facebook and Twitter social graphs with a methodology inspired by [26]. We start with a random set of a few users as seeds, traverse the social graph via breadth first search, until reaching the target number of nodes; our returned sample includes all edges among the nodes. The size of our samples is 1K or 10K, i.e., $|V| \approx$ 1K and $|T| \approx$ 1K, or $|V| \approx$ 10K and $|T| \approx$ 10K. Fig. 7 shows the complementary cumulative distribution function (CCDF) of follower/followee counts for both the original Facebook and Twitter datasets, and for our extracted datasets in the inner plot. The plots indicate that the original dataset properties were retained in the extracted sample. We also observe from Fig. 7 that the Twitter data has more correlation than the Facebook data. We denote the sampled instances by FB 1K, FB 10K, TW 1K, and TW 10K, respectively.

Churn traces: We use Google cluster data [1], which consists of traces from an 12K-machine cell over about a month-long period. We take two event types from the machine event table:

- ADD: a machine became available in the cluster.
- REMOVE: a machine was removed from the cluster.

We randomly sampled 1K and 10K unique machine IDs from the start of the Google cluster trace and then generated *Join* churn events via ADD and *Leave* ones via REMOVE.

6.2 Experiment plan

We first concentrate on OMen alone and tune its design parameters for the best performance. We compare the two algorithms for building backups and test local view selection with different coverage factors. Since failure handling is coordinated across the backups of the failed node, the size of the backup set is important for decentralization [R3]. We also investigate how evenly the load of serving as a backup is distributed across the nodes [R4], and how the selection of backups affects the overlay quality [R1].

We next evaluate all systems listed in Table 3 using the following optimization metrics: (1) the node degrees that [R1] emphasizes; (2) the overlay repair time [R2], i.e., the time $\tau_e(chev) - \tau_s(chev)$ that the entire churn handling procedure takes for a churn event *chev*. We also consider a few additional metrics, including overlay diameters [R1] and communication cost [R4].

6.3 Building backups greedily versus by primal-dual

We compare two algorithms for building backups (see §4.2). To eliminate other factors in the design space, we deploy two OMen-G instances with `buildBackupsGreedily()` and `buildBackupsByPD()`, respectively. For both instances, we set $\mathbb{L} = 1$ and $\lambda = 3$, initiate the same TCOs, and feed them with identical churn traces.

We derive a directed graph from the backup set for each node: there is an arc $v \rightsquigarrow w$ if $w \in B(v)$. Thus, the out-degree of node v is $|B(v)|$, and the in-degree of node v is $|\{u|v \in B(u)\}|$, the number of nodes that choose v as a backup.

Fig. 3 and Fig. 4 show the backup in/out degrees under Unif. Both `buildBackupsGreedily()` and `buildBackupsByPD()` functions

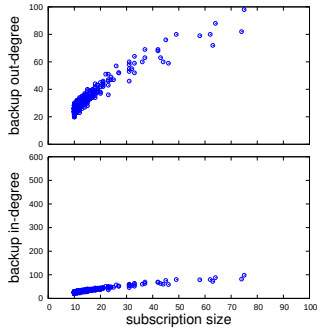


Figure 3: buildBackupsByPD()

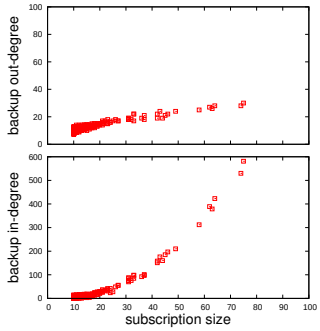


Figure 4: buildBackupsGreedily()

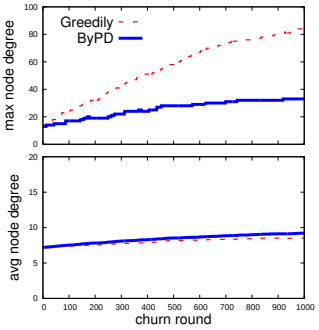


Figure 5: TCO node degrees

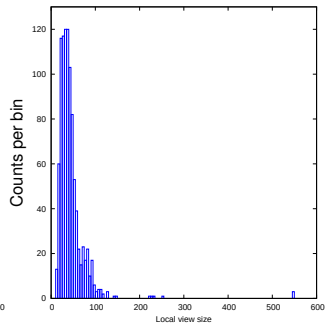


Figure 6: Local view - FB 1K

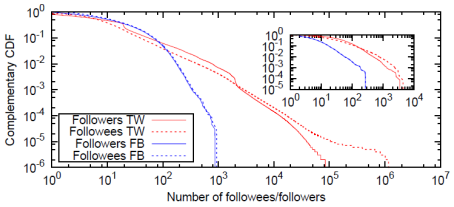


Figure 7: CCDF of followers and followees: Twitter (41.7M users) and Facebook (3M users). Inner plot: 10K-user sample.

produce small-sized backup sets compared to the complete node set V_0: 2.8% of $|V_0|$ for buildBackupsByPD() and 1.1% of $|V_0|$ for buildBackupsGreedily(), on average, across all nodes, respectively. In general, the backup degree is linearly proportional to the subscription size for both algorithms. However, the distributions of the in-degrees differ considerably. The in-degrees produced by buildBackupsGreedily() are skewed and follow an exponential shape: 42.9% of all nodes have in-degrees smaller than 5, while the highest in-degree of a single bulk subscriber peaks at 581. At the same time, the distribution of the in-degrees produced by buildBackupsByPD() is well-balanced: The lowest in-degree is 21 and the highest is 59. Note that buildBackupsGreedily() causes even more severe unfairness under skewed distributions: the fraction of nodes with in-degrees smaller than 5 is 89.8% and 95.2% under Zipf and Expo; nodes with in-degrees over 100 constitute about 1% yet serve as a backup for 35.8%, 50%, and 66.9% of all nodes, under Unif, Zipf, and Expo, respectively. These results confirm our observations in §4.2 about the effect of greediness on the primary assignment and fairness introduced by randomness in the primal-dual scheme.

We also evaluate how the two backup construction algorithms impact the overlay properties over a sequence of churn events. As shown in Fig. 5, both the maximum and average node degrees increase as the TCO instance evolves with node churn. However, overlay quality for OMen-G utilizing buildBackupsGreedily() degrades noticeably, i.e., at churn round 1000, the maximum node degree becomes 84. On the other hand, when the backups are built by primal-dual, both the maximum and average degrees show a steadily low growth rate.

Results presented in Fig. 3, 4 and 5 substantiate our choice of buildBackupsByPD() over buildBackupsGreedily() with regard to overlay quality [R1], decentralization [R3], and fairness [R4]. For the rest of the evaluation, we choose buildBackupsByPD() for selectView() (see Alg. 4).

6.4 Local view, backup set, and coverage factor

We explore the impact of local view selection with different coverage factors on the output and performance of OMen. We evaluate OMen with different values of the coverage factor ($\lambda = 0, ..., 9$).

Fig. 8(a) shows that the average local view in OMen is fairly small as compared to the overall size of the network, which meets the requirement of decentralization [R3]. Under FB 1K, the local view is 21.02 with $\lambda = 1$, 40.49 with $\lambda = 3$, 48.40 with $\lambda = 5$, and 57.59 with $\lambda = 9$, on average. The average local view also decreases over time, and the majority of nodes keeps a small local view under all instances. This is because, OMen evolves with constant epidemic exchange, and the local view converges rapidly with more balanced load distribution. Under all instances, after the first 100 rounds of gossiping, more than 95% of the nodes in $\text{OMen}|_{\lambda=5}$ have in their local view fewer than 5% of all nodes. This shows that OMen achieves good load balancing in local view maintenance.

Fig. 8(b) shows that, as λ increases, D_{OMen} decreases, and its growth rate with respect to the churn round decreases. The differences in the maximum node degrees and their growth rates also decrease with successive coverage factors. Under FB 1K, at the end of the Google cluster churn trace, $D_{\text{OMen}}|_{\lambda=0} = 333$, $D_{\text{OMen}}|_{\lambda=1} = 313$, $D_{\text{OMen}}|_{\lambda=3} = 302$, $D_{\text{OMen}}|_{\lambda=5} = 283$, $D_{\text{OMen}}|_{\lambda=7} = 277$, and $D_{\text{OMen}}|_{\lambda=9} = 274$. When $\lambda \geq 5$, the gap of the maximum node degrees with successive λ values is insignificant. The average node degrees follow the same trends with respect to the coverage factor, and the difference among successive λ values is even more trivial.

Fig. 6 plots a snapshot of the histogram for the local view sizes among all nodes, where we set $\lambda = 5$. OMen attains good balance in terms of the sizes of the local views [R4]: 97.4% of all nodes have ≤ 80 entries in their local view.

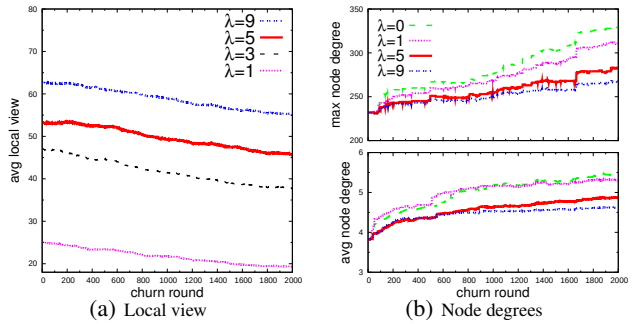
These experiments demonstrate that our selectView() method achieves good load balance in terms of local view maintenance across all nodes and confirm the validity of choosing a relatively small coverage factor (see §4.2). We fix $\lambda = 5$ for OMen in the rest of §6, which demonstrates the scalability, efficiency and robustness.

6.5 Overlay node degrees

We focus on the node degrees, one of the most important overlay qualities that [R1] defines. Fig. 9 compares the node degrees produced by different algorithms and protocols as the instances evolve based on the Google churn trace, where we construct the initial TCO by running LowODA from scratch. We do not plot lines for OMen-G or LowODA-Inc, because OMen-G lies close to OMen and LowODA-Inc lies close to LowODA-Re, e.g., $D_{\text{OMen}} \leq 1.04 \cdot D_{\text{OMen-G}}$, and $\bar{d}_{\text{OMen}} - \bar{d}_{\text{OMen-G}} \leq 0.76$, on average under FB 1K .

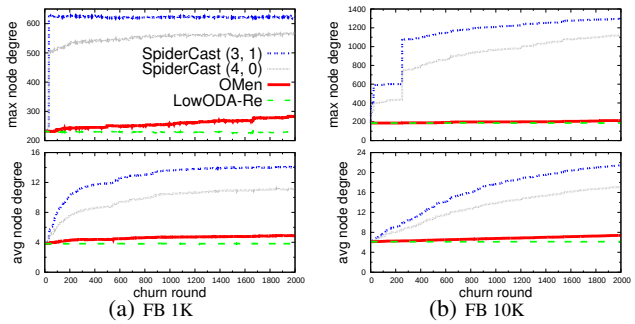
First, OMen outputs similar maximum and average node degrees as compared to LowODA-Re. For example, $D_{\text{OMen}} \leq 1.15 \cdot D_{\text{LowODA-Re}}$, $\bar{d}_{\text{OMen}} - \bar{d}_{\text{LowODA-Re}} \leq 1.12$, on average over the entire churn sequence under FB.

Second, the node degrees of OMen degrade *slowly* like a step function along the churn rounds. From churn round 1 to 2000 under FB 1K, D_{OMen} increases from 232 to 283, a degradation rate of 0.025 per churn round. This rate becomes even *slower* as the input



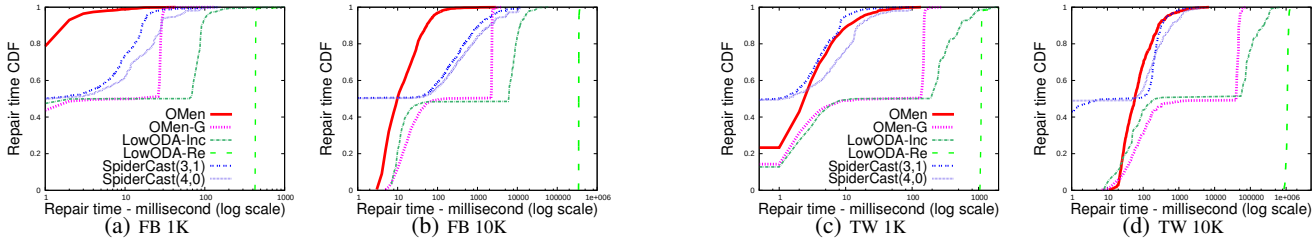
(a) Local view (b) Node degrees

Figure 8: OMen with λ's - FB 1K



(a) FB 1K (b) FB 10K

Figure 9: Node degrees under churn



(a) FB 1K (b) FB 10K (c) TW 1K (d) TW 10K

Figure 10: Repair time

instance scales up, which drops to 0.013 under FB 10K.

Third, SpiderCast degrades fast in the first 200 churn rounds, and the node degrees stay a number of times higher than those of OMen and LowODA. The gap expands as the instances scale up: $D_{\text{SpiderCast}(3,1)} - D_{\text{OMen}}$ is 355 under FB 1K and 1050 under FB 10K, respectively on average. SpiderCast$(3,1)$ generates more edges than SpiderCast$(4,0)$ due to random coverage, which leads to a higher probability to attain TCO [7].

6.6 Overlay repair time

We look at the overlay repair time in presence of churn events [R2]. Fig. 10 depicts *cumulative distribution functions* (CDFs) for the repair times measured for different rounds of a churn sequence. We plot CDFs for different algorithms, which were tested by injecting the same sequence of churn events from the Google cluster trace. As explained in §5, the repair time for OMen is measured as the life cycle duration for each churn event $chev$, i.e., $\tau_e(chev) - \tau_s(chev)$; it is dominated by the cost of overlay repair algorithms in Alg. 1, especially since the local view layer only requires a handful of gossip rounds to get back to the ready state upon each simple churn event.

In Fig. 10, LowODA-Re runs considerably slower than other dynamic algorithms, because LowODA-Re tears down existing links and reconstructs the TCO from scratch at each churn round. The runtime costs of LowODA-Inc and OMen-G are of the same order of magnitude. OMen and SpiderCast improve the time efficiency vastly: OMen is 4.67% that of OMen-G on average across all instances, thanks to local operations rather than global computation.

The speedup of OMen as compared to static LowODA-Re is more profound when the size of instance increases from 1K up to 10K. The running time ratio of OMen against LowODA-Re is around 0.20% under 1K and 0.01% under 10K, on average. This demonstrates the scalability of OMen with respect to the number of nodes, the number of topics, and the subscription sizes.

OMen, SpiderCast and LowODA-Inc require more time to dynamically repair the TCO under leaves than under joins. This can be explained by the number of nodes involved in the repair phase. Different magnitudes of time costs between handling joins and leaves form clear horizontal lines around 50% in the CDFs of OMen-G, SpiderCast, and LowODA-Inc. Meanwhile, the shape of running time CDF for OMen exhibits more smoothness and robustness across all churn rounds. One reason why OMen performs so well is that the backup set is being computed proactively in the background, while the repair protocol in the other systems is entirely reactive. This shows that OMen achieves balanced load distribution and fairness among all nodes in the network over a sequence of churn events.

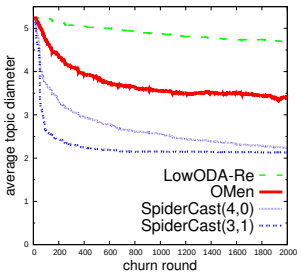


Figure 11: Diameter - TW 1K

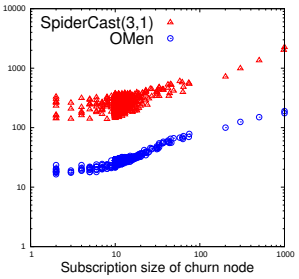


Figure 12: Comm - TW 1K

6.7 Topic diameters

We look at *topic diameters*, another important metric in [R1]. Given $TCO(V,T,I,E)$, the topic diameter for $t \in T$ is $diam^{(t)} = diam(G^{(t)})$, where $diam(G^{(t)})$ is the maximum shortest distance between any two nodes in $G^{(t)}$. We denote by $Diam$ and $\overline{diam}$ the maximum and average topic diameter across all topics.

Fig. 11 depicts the average topic diameters produced by different algorithms and protocols over the Google churn sequence under TW 1K. All systems start with the same topic diameters, which tend to decrease as the overlays evolve with the churn, because of the increasing overlay edges. Generally, topic diameters are inversely proportional to node degrees: SpiderCast has the lowest topic diameters, and LowODA-Re has the highest topic diameters. Topic diameters of OMen are slightly higher than those of SpiderCast: $Diam_{\text{OMen}} - Diam_{\text{SpiderCast}(3,1)} = 3.67$, and $\overline{diam}_{\text{OMen}} - \overline{diam}_{\text{SpiderCast}(3,1)} = 1.45$, on average.

6.8 Communication overhead

To evaluate the communication overhead for overlay maintenance [R4], we count messages sent and received by all nodes at each churn round. We rule out the communication overhead incurred by the failure detector, which is identical for all systems.

The communication overhead of OMen includes: (1) coordination messages for TCO recovery, e.g. handling concurrent churn events; (2) periodic message exchange for local view maintenance; and (3) notifications to establish links with new neighbors. Meanwhile, since SpiderCast is pure peer-to-peer, its communication overhead only contains (2) and (3) in the above list of OMen.

In Fig. 12, SpiderCast incurs significantly higher communication overhead as compared to OMen. The main reason is that SpiderCast has much higher node degrees, and every churn event impacts more nodes in SpiderCast.

Fig. 12 also shows that the systems require more messages when the churn node has a bigger subscription size. This comes from the fact that the bulk nodes with a large number of subscriptions (1) have more TCO neighbors and (2) are more likely to serve in the backup sets for other nodes.

7 Related work

7.1 Pub/sub overlays

A large body of work designs overlay topologies for distributed pub/sub systems to optimize the network traffic (e.g., [6, 21]). The TCO property is required in [3, 7, 22, 25, 29], while all approaches aim to reduce intermediate overlay hops for message delivery.

On the one hand, aiming to achieve topic-connectivity while minimizing node degrees has been explored algorithmically in [5, 6, 21], where different optimization goals led to a number of algorithms. These algorithms have proven approximation ratios and serve as building blocks and comparison baselines for developing other approaches. All of these algorithms are *static* by design: they assume centralized operation, require global knowledge, and perform overlay construction from scratch, as opposed to incremental adaptation. These innate static properties are undesirable in dynamic environments. Furthermore, these algorithms suffer from high time complexity, and it is impractical to rerun them each time a single node joins or leaves.

On the other hand, systems like [7, 22, 11, 29] build the TCO in a decentralized manner. These systems implement non-coordinated decentralized overlay construction protocols such that each node decides upon its own neighbors independently. The protocols are fast and can operate with only local and partial knowledge. However, these heuristics do not provide any theoretical guarantees for the node degrees. In practice, the node degrees are usually several times higher than the bounds provided by the static, centralized baselines in [5, 6, 21].

A few pub/sub systems [4, 26] build degree-bounded overlays while compromising on the TCO property.

OMen adopts a hybrid approach that resides between the centralized algorithms and decentralized protocols. We design effective methods to tune the collective coordination among decentralized settings. Our system achieves both overlay quality and time efficiency, whereas previous work only optimize one metric but perform poorly with regard to the other.

Pub/sub overlays are fundamentally different from the canonical distributed overlay networks, e.g., small-world networks [15]. In pub/sub, additional semantic information is associated with each individual node, such as topic interests. This semantic information radically affects the overlay structure. As a result, pub/sub overlay design poses unique challenges: (1) the need to preserve TCO integrity upon overlay mending; (2) scalability in the number of topics (i.e., diverse interests) and the subscription size.

7.2 Backup nodes

OMen is the first distributed system that proposes backup nodes for fault-tolerant design of pub/sub. The notion of backup nodes is not new, but it is only used for the simplest case of peer-to-peer overlay recovery. HyParView [19] maintains a smaller *active* view and a larger *passive* view. In case of failures, HyParView promotes the backup nodes in the passive view to the active view. However, HyParView only supports broadcast and is not directly applicable to pub/sub. Actually, the idea of backup nodes has never been applied in pub/sub overlay repair, mainly because pub/sub poses special challenges for defining and building backup sets. In particular, OMen has to address the requirements of topic coverage and individual commitment for backup sets, which HyParView (or other existing systems) does not consider.

7.3 Gossiping

We choose gossiping for building and maintaining local views, including backup sets, across all nodes in a distributed manner (see §3.2 and §4.3). Although we employ the epidemic framework of T-Man, OMen has several distinguishing characteristics compared to typical T-Man applications:

1. T-Man normally defines the target topology by a single *ranking function*, that each node can apply to order any set of other nodes according to preference for choosing them into the local view. Many systems define the T-Man ranking functions through a distance function that defines a metric space over the set of nodes, considering geographical location, semantic description of stored content, storage capacity, etc. However, in OMen, the local view selection at one node needs to consider the local view selection at other nodes, which is difficult to capture by a metric-based distance function.
2. The local view length is non-uniform across all nodes. The inherent skewness of pub/sub workloads may lead to inevitable bias in the local view. This bias is undesirable and can have a detrimental effect on the performance of OMen, including both overlay quality [R1] and fairness [R4] (see Fig. 4 and Fig. 5 in §6). Despite the non-uniform view lengths, we strive to limit the bias and retain sufficient amount of randomness in the local view (see §4.2). The results are evident in our empirical evaluation (see Fig. 6 in §6).

8 Conclusions

We presented a fully dynamic system, OMen, to construct and maintain low node degree TCOs for pub/sub systems under churn. We demonstrated that OMen combines the advantages from centralized algorithms and decentralized protocols, under large-scale pub/sub workloads extracted from Twitter and Facebook and real-world churn traces released by Google: (i) both the maximum and average node degrees remain close to those of the centralized algorithms and (ii) the running time efficiency is of the same order of magnitude as for the decentralized protocols.

9 References

[1] Google Cluster Data. http://code.google.com/p/googleclusterdata.

[2] IBM IoT Foundation. http://internetofthings.ibmcloud.com/.

[3] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera: Topic-based event routing for peer-to-peer architectures. In *DEBS '07*.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level

multicast infrastructure. *JSAC*, 2002.

[5] C. Chen, R. Vitenberg, and H.-A. Jacobsen. Scaling construction of low fan-out overlays for topic-based publish/subscribe systems. In *ICDCS'11*.

[6] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing scalable overlays for pub-sub with many topics: Problems, algorithms, and evaluation. Podc'07.

[7] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS'07*.

[8] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 1979.

[9] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 2008.

[10] S. El-Ansary, L. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. In *Peer-to-Peer Systems II*. 2003.

[11] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed. Magnet: practical subscription clustering for internet-scale publish/subscribe. Debs'10.

[12] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 1982.

[13] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Middleware '04*.

[14] M. Jelasity, A. Montresor, and O. Babaoglu. T-man: Gossip-based fast overlay topology construction. *Computer Networks*, 2009.

[15] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *STOC'00*.

[16] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? WWW'10.

[17] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Operating System Review*, 2010.

[18] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, pages 133–169, 1998.

[19] J. Leitao, J. Pereira, and L. Rodrigues. HyParView: A membership protocol for reliable gossip-based broadcast. In *DSN '07*.

[20] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. IMC'05.

[21] M. Onus and A. W. Richa. Parameterized maximum and average degree approximation in topic-based publish-subscribe overlay network design. ICDCS'10.

[22] M. Matos, P. Felber, R. Oliveira, J. O. Pereira, and E. Riviere. Scaling up publish/subscribe overlays using interest correlation for link sharing. *TPDS*, 2013.

[23] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *P2P'09*.

[24] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *USENIX ATC'14*.

[25] J. A. Patel, E. Rivière, I. Gupta, and A.-M. Kermarrec. Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. *Computer Networks*, 53(13):2304–2320, 2009.

[26] 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. IPDPS'11.

[27] J. Reumann. Pub/Sub at Google. Lecture & Personal Communications at EuroSys & CANOE Summer School, Oslo, Norway, Aug 2009.

[28] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *ATEC '04*.

[29] V. Setty, M. van Steen, R. Vitenberg, and S. Voulgaris. Poldercast: Fast, robus, and scalable architecture for p2p topic-based pub/sub. Middleware'12.

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

[31] G. Urdaneta, G. Pierre, and M. V. Steen. Towards a fully decentralized and collaborative hosting infrastructure for Wikipedia. In *WikiSym'08*.

[32] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *EuroSys '09*.