Reinforce Your Overlay with Shadows: Efficient Dynamic Maintenance of Robust Low Fan-out Overlays for Topic-based Publish/Subscribe under Churn

Chen Chen
University of Toronto
chen@msrg.utoronto.ca

Hans-Arno Jacobsen
University of Toronto
jacobsen@eecg.toronto.edu

Roman Vitenberg
University of Oslo
romanvi@ifi.uio.no

Abstract

Overlay network design for topic-based publish/subscribe systems is of primary importance because the overlay directly impacts the system's performance. Determining a low fan-out *topic-connected overlay* (TCO) is a fundamental problem.

Existing algorithms for constructing TCOs with provably low fan-out build the overlays from scratch. In this paper, we propose the first fully dynamic algorithms for efficiently maintaining TCO in presence of node churn such that both the average and maximum node degrees stay provably low. The main challenge of dynamic maintenance is to efficiently overcome departure of nodes central to topic-connectivity. This is attained by maintaining sets of *shadow nodes* so that all links adjacent to a failed node can quickly be replaced by adding links among shadow nodes.

Compared to constructing TCOs from scratch, the proposed algorithms exhibit two important advantages: When a node joins or leaves, topic connectivity is restored much faster, and changes to the overlay are incremental so that existing links do not need to be torn down. We corroborate these advantages by extensive evaluations on typical workloads with up to 2 000 nodes and 200 topics under 1 000 rounds of churn.

Keywords

Topic-based Publish/subscribe; Churn; Overlay;

I. Introduction

Publish/subscribe (pub/sub) systems constitute an attractive choice as communication paradigm and messaging substrate for building large-scale distributed systems. Many real-world applications are using pub/sub for message dissemination, such as application integration across data centers [1], financial data dissemination [2], RSS feed aggregation, filtering, and distribution [3]. Besides these applications, many standards have adopted pub/sub as part of their specifications, including WS Notifications, WS Eventing, the OMG's Real-time Data Dissemination Service, and the Active Message Queuing Protocol.

In pub/sub, subscribers express their interests in receiving messages through subscription messages and publishers disseminate publication messages. 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 on topics of interest.

In a distributed pub/sub system, so called pub/sub brokers, often connected in a federated manner as an application-level overlay network, efficiently route publication messages from data sources to sinks. The overlay properties directly impacts the system's performance and the message routing cost. Constructing a high-quality broker overlay is a key challenge and fundamental problem for distributed pub/sub systems that has received attention both in industry [1], [4] and academia [5], [6], [7], [8], [9], [10].

In [5], the notion of topic connectivity was defined, which informally speaking means that all nodes (i.e., pub/sub brokers) interested in the same topic are organized in a connected dissemination sub-overlay. This property ensures that nodes not interested in a topic never need to contribute to disseminating information on that topic. Publication routing atop such overlays saves bandwidth and computational resources otherwise wasted on forwarding messages of no interest to the node. It also results in smaller routing tables.

Apart from topic-connectivity, it is imperative for an overlay network to have a low node degree. It costs a lot of resources to maintain adjacent links for a high-degree node (i.e., monitor the links and the neighbors [5], [7]). Besides, for a typical pub/sub system, each link would have to accommodate a number of protocols, service components, message queues and so on. While overlay designs for different applications might be principally different, they all share the strive for maintaining bounded node degrees, whether in DHTs [11], wireless networks [12], or for survivable network design [13].

Unfortunately, the properties of topic-connectivity and low node degree are at odds with each other. Intuitively, a sparse overlay is unlikely to be topic-connected while a dense overlay is sub-optimal with respect to the node degree. This trade-off has been explored in a number of approaches for constructing a topic-connected overlay while minimizing the average or maximum node degree [5], [6], [7], [10]. Notably, all these state-of-the-art algorithms are designed to construct the overlay from scratch.

To the best of our knowledge, this work is the first to consider the problem of dynamically maintaining a pub/sub overlay in presence of churn such that topic-connectivity is guaranteed to be quickly restored and the degrees stay provably low. While it is possible in principle to rebuild the overlay from scratch after each node join or leave, this approach is not deemed

practical: The above state-of-the-art algorithms have a high running time cost of $O(|V|^2|T|)$ where $|V|$ is the number of nodes and $|T|$ is the number of topics. Furthermore, tearing down all existing links and establishing potentially different links anew is also prohibitively expensive.

The main challenge of dynamic maintenance is to overcome the departure of a node central to topic-connectivity. In this situation, additional edges need to be created in order to mend the overlay and restore topic-connectivity. If the protocol considers the entire set of potential edges that can be added, its running time will be problematic. If the protocol only considers a small subset of edges for addition, those edges may turn out suboptimal thereby significantly increasing the degrees.

To overcome this challenge our solution builds and maintains a set of backup nodes for each node in the overlay. This set is constructed when a node joins and incrementally updated as needed, e.g., when a backup node leaves. When a node v leaves, we calculate the *shadow set* of v, which is the union of v's neighbors and v's backup set. Only edges between the nodes in shadow set are considered for addition towards restoring topic-connectivity. We present and analyze alternative ways to calculate the backup set in such a way that (a) the incremental computation needed for restoring topic-connectivity is fast, (b) the degrees are kept provably low, and (c) the backup sets can be efficiently updated. In particular, these requirements imply that a node should not be assigned as a backup for a high number of other nodes. More generally, backups for different nodes should be distributed as uniformly as possible for a given input.

We consider the performance of the proposed dynamic algorithms and provide analytical bounds for the running time as well as maximum and average degrees. Specifically, we show that under assumptions that hold for typical pub/sub workloads, the incremental dynamic algorithms run much faster while the produced degrees remain close to those of the state-of-the-art static algorithms. Apart from the theoretical analysis, we conduct comprehensive experiments under a variety of characteristic pub/sub workloads with up to 2 000 nodes and 200 topics under 1 000 rounds of churn. The results show that our dynamic algorithms run hundreds of times faster and reshuffle hundreds of times fewer edges at each churn round compared to the static baseline algorithms, at the cost of an insignificant increase in maximum and average degrees.

II. Related Work

Significant body of research in distributed pub/sub systems has been considering construction of the underlying overlay topology so that network traffic is minimized (e.g [5], [6], [8], [9], [10], [7], [14], [15], [16]). *Topic-connectivity* is a required property in [17], [18], [16]. It is an implicit requirement in [19], [20], [9], [21], [14], which aims to reduce the number of intermediate overlay hops for a message using a variety of techniques.

The goal of constructing a *topic-connected overlay* while minimizing node degrees, has resulted in the formulation of various research problems: MinAvg-TCO for average degree [5], MinMax-TCO for maximum degree [6], and Low-TCO for both average degree and maximum degree simultaneously [7]. Greedy approximation algorithms have been proposed for these problems [5], [6], [7]. Chen *et al.* [10], [15] present a divide-and-conquer overlay design method, which significantly reduces the time and space complexity of constructing a low fan-out *topic-connected overlay*. All of these algorithms are *static* and construct the overlay from scratch.

While resilience to churn has been considered in a large number of decentralized architectures for pub/sub, only a few of those ([18], [16]) focus on providing topic-connectivity and low fan-out. Both [18] and [16] achieve topic-connectivity with high probability in a failure-free run. Their empirical evaluation also shows that the resulting overlay tends to have moderate degrees. In contrast, our work provides a deterministic guarantee of topic-connectivity and provable bounds on the degrees due to adopting a principally different less decentralized implementation approach.

Patent [22] proposes a simple *dynamic* strategy of applying the algorithm in [5] towards mending topic-connectivity of the overlay. This approach has a high runtime cost similar to that of the static algorithm of [5]. Moreover, [22] has no guarantee to bound the maximum node degree in the produced overlay.

III. Background

Let V be the set of nodes and T be the set of topics. Each node $v \in V$ is interested in a set of topics, denoted as $v.topics \in T$. We say that a node v is interested in topic t if and only if $t \in v.topics$; we also say that v subscribes to t, or t is subscribed by v. $|v.topics|$ is called the *subscription size* of node v. Further, the subset of nodes that are interested in topic $t \in T$ is denoted by $t.nodes$, i.e., $t.nodes = \{v | t \in T \wedge t \in v.topics\}$.

An overlay network $G(V, E)$ is an undirected graph over the node set V with the edge set $E \subseteq V \times V$. Given an overlay network $G(V, E)$ and a topic $t \in T$, we say that a sub-graph $G_t(V_t, E_t)$ of G is *induced* by t if $V_t = \{v \in V | t \in v.topics\}$ and $E_t = \{(v, w) \in E | v \in V_t \wedge w \in V_t\}$. An overlay G is called *topic-connected* if for each topic $t \in T$, the sub-graph G_t of G induced by t contains at most one *topic-connected component* (*TC-component*). A *topic-connected overlay* (TCO) for given V, T, and E is denoted as $TCO(V, T, E)$.

In this paper we focus on exploring TCO problems in *dynamic* environments. To exemplify our algorithm design and analysis under churn, we target at the optimizing the *static* properties of Low-TCO, which is formally defined as follows:

In [7], the authors defined the *static* problem of constructing a TCO from scratch while minimizing both maximum and average node degrees.

Definition 1. *Low-TCO*(V, T)*: Given a set of nodes* V *in which each node subscribes to a subset of topics from* T*, construct a* topic-connected overlay $TCO(V, T, E)$ *while minimizing both maximum and average node degrees.*

Onus *et al.* [7] proposed the Low-ODA algorithm for Low-TCO. Low-ODA is specified in Alg. 1 and operates in a greedy manner as follows: It starts with an empty set of edges and iteratively adds carefully selected edges one by one until topic-connectivity is attained. The edge selection rule, instantiated by **findLowEdge()** (line 5 of Alg. 2), is based on a combination of two criteria: node degree and *edge contribution*, which is defined as reduction in the number of topic-connected components caused by the addition of the edge to the current overlay. Edge contribution for an edge e is denoted as $contrib(e)$. Function **findLowEdge()** uses a parameter ρ to tread the balance between average and maximum node degree, and it makes a weighed selection between two candidate edges:

- e_1 s.t. $contrib(e_1)$ is maximum among E_{pot};
- e_2 s.t. $contrib(e_2)$ is maximum among the subset of E_{pot} in which all edges increase maximum degree of $G(V, E_{new} \cup E_{new})$ minimally.

If $contrib(e_1)$ is greater than $\rho \cdot contrib(e_2)$, edge e_1 is added; otherwise e_2 is added.

Algorithm 1 Low-ODA for *Static* Low-TCO

Low-ODA$(I(V, T))$
Input: $I(V, T)$
Output: A topic-connected overlay $TCO(V, T, E_{new})$

1: $E_{new} \leftarrow$ **constructLowEdges**$(V, T, \emptyset, V \times V)$
2: return $TCO(V, T, E_{new})$

Algorithm 2 Overlay Construction for Low-ODA

constructLowEdges(V, T, E_{cur}, E_{pot})
Input: V, T, E_{cur}, E_{pot}
// E_{cur}: Set of current edges that exist in the overlay
// E_{pot}: Set of potential edges that can be added
Output: Edge set E_{new} that combined with E_{cur}, forms a TCO

1: $E_{new} \leftarrow \emptyset$
2: **for all** $e(v_1, v_2) \in E_{pot}$ **do**
3: $contrib(e) \leftarrow |\{t \in T | t \in v_1.topics \wedge t \in v_2.topics \wedge$ v_1, v_2 belong to different *TC-components* for t in $G(V, E_{cur})\}|$
4: **while** $G(V, E_{new} \cup E_{cur})$ is not topic-connected **do**
5: $e \leftarrow$ **findLowEdge**(ρ)
6: $E_{new} \leftarrow E_{new} \cup \{e\}$
7: $E_{pot} \leftarrow E_{pot} - \{e\}$
8: **for all** $e(v_1, v_2) \in E_{pot}$ **do**
9: $contrib(e) \leftarrow$ update the contribution of a potential edge e as the reduction on the number of *TC-components* which would result from the addition of e to $G(V, E_{new} \cup E_{cur})$
10: return E_{new}

IV. Dynamic Topic-Connected Overlay Problems

In the *dynamic* TCO problem, we are given a set T of topics, initial set V_0 of nodes and their subscriptions, and an already constructed TCO for T and V_0 with low degrees. We are further given a sequence $\Delta = (\delta_1, \delta_2, ..., \delta_k, ...)$ of churn rounds, each round incurring a single change to the set V of nodes and their subscriptions. Our goal is to incrementally maintain the TCO in presence of churn:

Definition 2. *Dyn-Low-TCO*$(TCO_0(V_0, T, E_0), \Delta)$*:*
Given a solution $TCO_0(V_0, T, E_0)$ *for an initial* Low-TCO *instance* $I_0(V_0, T)$*, and a* sequence Δ *of churn rounds, dynamically maintain* TCO_k *with minimal maximum and average node degrees for each round* δ_k *of churn.*

Various types of churn can be defined for Δ in Dyn-Low-TCO, such as node joining or leaving, changes in the subscriptions, as well as combined changes. In this paper, we focus on joins and leaves and assume that at each $\delta_k \in \Delta$ only one node joins or leaves the overlay. Observe that this type of churn is more challenging to handle compared to minor changes in the subscriptions. Besides, algorithms for this basic type of churn can be extended to other types, like churn concerning batch alterations.

Dyn-Low-TCO can be tackled by applying the *static* Low-ODA to re-construct the TCO "from scratch" for each new instance at churn round δ_k. However, it is certainly better from the practical standpoint to preserve existing edges and only incrementally add edges as necessary. This is because establishing a new edge is a relatively costly operation that entails construction of new routing tables. Furthermore, incremental computation of additional edges is more efficient than re-computation of the entire overlay.

V. Dynamic algorithms for TCO problems

A. Static *Low-ODA* as a building block

In this section, we analyze the greedy Low-ODA in greater detail and derive new results for the algorithm when running on a partially constructed overlay. *Static* Low-ODA is employed as a building block for our *dynamic* algorithms and serves as baseline in our experimentation.

Given a Low-TCO instance $I(V,T)$, let us denote $D_{MIN}(V,T)$ (or $D_{MIN}(I)$) as the minimum possible maximum node degree of any TCO for instance I, and $d_{MIN}(V,T)$ (or $d_{MIN}(I)$) as the minimum possible average degree of any TCO for I. Furthermore, we denote the maximum degree of a graph $G(V,E)$ by $D(V,E)$, and the average degree by $d(V,E)$. More specifically,

$$\begin{aligned} D_{MIN}(V,T) &= \min_{TCO(V,T,E)} D(V,E) \\ d_{MIN}(V,T) &= \min_{TCO(V,T,E)} d(V,E) \end{aligned}$$

In [7], Alg. 2 is the only entry point for Alg. 1. This means that E_{cur} is always equal to $\emptyset$ and E_{pot} to $V \times V$ upon the invocation of Alg. 2. For Dyn-Low-TCO, we need to apply the greedy algorithm on a partially constructed overlay. Therefore, we have to extend the analysis of Low-ODA for the case when $E_{cur} \neq \emptyset$ and $E_{pot} = (V \times V) \backslash E_{cur}$. Let E_{new} be the set of edges returned by Alg. 2, then the following lemma holds:

Lemma 1. *If invoked on (V,T,E_{cur},E_{pot}) such that $E_{pot} = (V \times V) \backslash E_{cur}$, and **findLowEdge()** is parameterized with ρ, then Alg. 2 has the following properties:*
(a) the running time is $O(|E_{pot}||T|) = O(|V|^2|T|)$,
(b) $G(V, E_{cur} \cup E_{new})$ is topic-connected *for instance $I(V,T)$,*
(c) the maximum node degree $D(V, E_{cur} \cup E_{new})$ is bounded by $O(D(V,E_{cur}) + \frac{|V|}{\rho} D_{MIN}(V,T) \cdot \log(|V||T|))$,
(d) the average node degree $d(V, E_{cur} \cup E_{new})$ is bounded by $O(d(V,E_{cur}) + \rho \cdot d_{MIN}(V,T) \cdot \log(|V||T|))$.

Proof: (a) The improvement of the running time from $O(|V|^4|T|)$ to $O(|V|^2|T|)$ was proven in Lemma 7 in [23].

(b) The correctness of Alg. 2 follows directly from the termination condition in line 4 of Alg. 2.

(c) The approximation ratio for maximum node degree stems from Theorem 2 in [7] and the proof is similar to the one for Lemma 3 in [15].

(d) The approximation ratio for average node degree follows exactly the same proof for Theorem 1 in [7]. ■

B. Naive Dynamic Algorithms for *Dyn-Low-TCO*

Given Alg. 2 and Lemma 1 for the *static* Low-TCO, we first consider a set of *Naive Dynamic Algorithms*, called NaiveDynAlg, for handling incremental and decremental churn. NaiveDynAlg is also based on a greedy heuristic giving rise to similar properties as for Alg. 2. We include the design and analysis of these algorithms below to illustrate their weaknesses that motivate the need for more advanced algorithms, presented thereafter.

Alg. 3 and Alg. 4 present NaiveDynAlg. The intuition behind these algorithms is that we need to determine a set of new edges E_{new}, that in conjunction with the current edge set E_{cur}, produces a TCO for the new instance. The algorithms start with a non-empty E_{cur} set and compute the potential edge set E_{pot} containing all possible edges. Then, they iteratively select edges one by one from E_{pot} using **findLowEdge()** until a new TCO is obtained. Note that only those of the topics that become disconnected as a result of churn need to be considered, i.e., $v'.topics \in T$.

Algorithm 3 Naive Algorithm for Single Node Join

constructOverlayOnJoinNaively($TCO_{cur}(V,T,E_{cur}), v'$)
Input: current overlay TCO_{cur}, and joining node v'
Output: $TCO_{new}(V',T,E_{new})$: new TCO that contains v'
1: $V' \leftarrow V \cup \{v'\}$
2: $E_{pot} \leftarrow \{e(v',u) | u \in V'\}$
3: $E_{new} \leftarrow$ **constructLowEdges**($V', v'.topics, E_{cur}, E_{pot}$)
4: return $TCO_{new}(V', T, E_{cur} \cup E_{new})$

Algorithm 4 Naive Algorithm for Single Node Leave

constructOverlayOnLeaveNaively($TCO_{cur}(V, T, E_{cur}), v'$**)**
Input: current overlay TCO_{cur}, and leaving node v'
Output: $TCO_{new}(V', T, E_{new})$: new TCO that excludes v'
1: $V' \leftarrow V - \{v'\}$
2: $E_{cur} \leftarrow E_{cur} - \{(v', u) \in E_{cur}\}$
3: $E_{pot} \leftarrow V' \times V' - E_{cur}$
4: $E_{new} \leftarrow$ **constructLowEdges**$(V', v'.topics, E_{cur}, E_{pot})$
5: return $TCO_{new}(V', T, E_{cur} \cup E_{new})$

Alg. 3 and Alg. 4 follow the design of the greedy algorithm for Low-TCO given in Sec. V-A. The correctness, running time and approximation ratio properties for our NaiveDynAlg can be established by adapting the results of Lemma 1.

Lemma 2. *Alg. 3 and Alg. 4 have the following properties:*
(a) the running time is $O(|E_{pot}||v'.topics|)$,
(b) $G(V', E_{cur} \cup E_{new})$ *is* topic-connected *for the new instance* I',
(c) the maximum node degree $D(V', E_{cur} \cup E_{new})$ *is bounded by* $O(D(V, E_{cur}) + \frac{|V'|}{\rho} D_{MIN}(V', v'.topics) \cdot \log(|V'||v'.topics|))$,
(d) the average node degree $d(V', E_{cur} \cup E_{new})$ *is bounded by* $O(d(V, E_{cur}) + \rho \cdot d_{MIN}(V', v'.topics) \cdot \log(|V'||v'.topics|))$.

If the base TCO_{cur} is constructed with Alg. 1, then Alg. 3 and Alg. 4 achieve the same approximation ratios as the greedy algorithm for the *static* case with regard to both maximum and average node degrees. However, the running time as given by Lemma 2 is not insignificant: $|E_{pot}| = O(|V'|)$ for node joining and $|E_{pot}| = O(|V'|^2)$ for node leaving, respectively. For node leaves, Alg. 4 takes $O(|V'|^2|T|)$ time, which is asymptotically identical to running *static* Alg. 1 that constructs the overlay from scratch. Besides, the *naive dynamic* overlay construction requires complete knowledge of all nodes. These shortcomings motivate the development of new algorithms.

C. Shadows and Primary-backup Strategy

In this section, we present the key concepts and design elements of new *dynamic* algorithms for Dyn-Low-TCO.

According to Lemma 2, the time complexity of NaiveDynAlg is $O(|E_{pot}||v'.topics|)$, in particular, $O(|V'||T|)$ for node join and $O(|V'|^2|T|)$ for node leave. The number of nodes, and thus the size of the potential edge set, turns out to be the principal factor for the time complexity of the TCO construction algorithms. Based on this observation, we try to reduce the size of the node set involved in the construction step.

When node v' is joining or leaving, in contrast to computing E_{pot} with V' as its domain (Line 2 of Alg. 3 and Line 3 of Alg. 4), it is possible to re-attain *topic-connectivity* by only adding links between a selected node subset. Formally speaking, we define the *shadow set* as the subset of nodes that are employed for overlay construction upon node churn.

Definition 3 (Shadow set). *Given an instance of Dyn-Low-TCO*$(TCO_0(V_0, T, E_0), \Delta)$*, the* shadow set *(or* shadows*) for a churn round* $\delta \in \Delta$*, is a subset of nodes from* V' *that are chosen in the overlay construction step for the new Low-TCO instance* $I'(V', T)$*, where* $V' = V \cup \{v'\}$ *for joining and* $V' = V \backslash \{v'\}$ *for leaving.*

The *shadow set* S can be regarded as a *sample* of nodes that is representative of specific characteristics for the entire node *population* V'. Taking *topic-connectivity* into account, it is always safe (but not necessarily efficient) to set *shadow set* as the complete node set V'. However, there might exist many other choices for the *shadow set* with much fewer nodes. In the case that node v' is leaving, one candidate for the *shadow set* is the *neighbor set* around the leaving node v', i.e., $v'.neighbors$. Observe that to re-attain *topic-connectivity* it is sufficient to add links among $v'.neighbors$ to the existing overlay. This can be done very efficiently since $v'.neighbors$ is usually much smaller than the complete node set V', but the node degrees of $v'.neighbors$ would usually be increased significantly in the output TCO. The trade-off between the runtime cost and the quality of the output TCO can be balanced by selecting the *shadow set* in between $v'.neighbors$ and V', which will be discussed in more detail in Section V-G.

In order to efficiently compute the *shadow set* upon churn, each node $v \in V$ continuously maintains a subset of nodes as its *backup set*, denoted as $v.backups$. A node $u \in v.backups$ is referred to as a *backup node* (or *backup*) for v, and $|v.backups|$ is called v's *backup degree*.

As illustrated in Fig. 1, when v' is leaving, the *backup set*, in conjunction with the *neighbor set*, forms the *shadow set* S to repair the "broken" TCO. The *backup set* for node $v \in V$ should preserve the following desirable properties:

1) each *backup node* u shares at least one *topic* with v:

$$u.topics \wedge v.topics \neq \emptyset, \forall u \in v.backups \tag{1}$$

2) all *topics* subscribed by v are covered by its *backups*:

$$v.topics \subseteq \bigcup_{u \in v.backups} u.topics \tag{2}$$

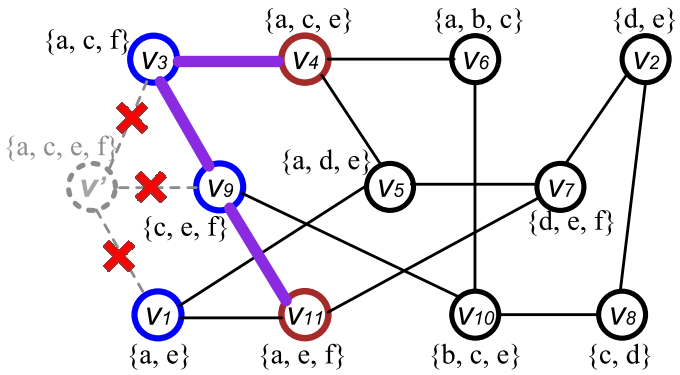

Figure 1. Node v' is leaving from the overlay, and a low fan-out TCO is restored by adding edges $\{(v_3, v_4), (v_9, v_{11}), (v_3, v_9)\}$ among the *shadow set* S, where $v'.backups = \{v_1, v_4, v_{11}\}$ and $S = v'.neighbors \cup v'.backups = \{v_3, v_9, v_1, v_4, v_{11}\}$.

Thanks to the notions of *shadow set* and *backup set*, we design a "primary-backup" strategy for handling node leaves:
Step I, build a subset of *backup nodes* for each node $v \in V$ in the base TCO in advance;
Step II, when a node v' leaves, **repair** the overlay "locally" among the *shadow set*, which is $v'.backups \cup v'.neighbors$;
Step III, update *backups* for those nodes that previously choose the departed node v' as a *backup*.

To allow efficient update of *backups* in the third step, each node $v \in V$ needs to keep a list of nodes for which it serves as *backup*. We call this list the *primary set* of node v, denoted as $v.primaries$:

$$v.primaries = \{u | v \in u.backups\},\ v \in V$$

A node $u \in v.primaries$ is referred to as one of v's *primary nodes* (or *primaries*), and $|v.primaries|$ is called v's *primary degree*.

Struct 5 encapsulates the node and its meta information required for our problem formulation: the set of subscribed topics, the neighbor set, the *backup set* and the *primary set*. Our new *dynamic* algorithms will be based on this data structure for nodes.

Struct 5 Extended Definition of NODESTRUCT

NODESTRUCT(v): extended encapsulation of a node and its field information
- $topics$: the subscribed topics of the node
- $neighbors$: the node's neighbors in the TCO
- $backups$: the node's *backups* in the TCO
- $primaries$: the set of nodes that choose this node as *backup*

Step I is specified in Alg. 6, which **builds** *backup set* and associated *primary set* for each node $v \in V$.

Algorithm 6 Step I: Build *Backups* for TCO Nodes

buildBackupsForTCO($TCO(V, T, E)$)
Input: TCO
Output: $backups$ (and $primaries$) built for each $v \in V$
1: **for all** node $v \in V$ **do**
2: **buildBackupsAt**(v)

The trade-off and design principles for building *backups* at each node is discussed in Section V-G. Before digging into **buildBackupsAt**(v) for $v \in V$ in Line 2 of Alg. 6, we consider **Steps II** and **III** and show how *shadows* can efficiently support TCO construction under node churn.

D. Dynamic Node Leave Algorithms with Shadows

Step II is accomplished by Alg. 7. The algorithm **repairs** TCO in two phases. First, it computes the *shadow set* S as the union of the *neighbor set* and the *backup set* which is determined at **Step I**, and then computes E_{pot} as the cartesian product of S. Second, it repairs the TCO by applying Alg. 2 on E_{pot} and considering only the subset of topics that are disconnected.

Step III is specified Alg. 8. After repairing TCO on v''s departure, the algorithm **updates** the fields of *backups* and *primaries* for affected nodes.

Algorithms at **Steps I** and **III** do not impact *topic-connectivity* of the overlay, and can operate off-line under node leaving. The responsiveness of TCO construction under node leaving only depends on the efficiency of **Step II**. Lemma 3 is established for the correctness, running time and approximation ratios for our Alg. 7 at **Step II**.

Algorithm 7 Step II: Repair TCO for Node Leave

repairOverlayOnLeaveByShadows($TCO_{cur}(V, T, E_{cur}), v'$)
Input: current overlay TCO_{cur}, and leaving node v'
Output: $TCO_{new}(V, T, E_{new})$: new TCO that excludes v'
1: $S \leftarrow v'.neighbors \cup v'.backups$
2: $E_{cur} \leftarrow E_{cur} - \{(v', n)\}$
3: $E_{pot} \leftarrow S \times S$
4: $E_{new} \leftarrow$ **constructLowEdges**$(S, v'.topics, E_{cur}, E_{pot})$
5: $V \leftarrow V' \backslash \{v'\}$
6: return $TCO_{new}(V', T, E_{cur} \cup E_{new})$

Algorithm 8 Step III: Update Backups after Node Leave

updateBackupsOnLeave(v')
Input: joining node v'
Output: $backups$ (and $primaries$) re-built for $u \in v'.primaries$
1: **for all** *primary node* $u \in v'.primaries$ **do**
2: **buildBackupsAt**(u)

Lemma 3. *Alg. 7 has the following properties for a node leave round:*
(a) the running time is $O(|S|^2|v'.topics|)$,
(b) $G(V', E_{cur} \cup E_{new})$ *is* topic-connected *for the new instance* I',
(c) the maximum node degree $D(V', E_{cur} \cup E_{new})$ *is bounded by* $O(D(V, E_{cur}) + \frac{|S|}{\rho} D_{MIN}(S, v'.topics) \cdot \log(|S||v'.topics|))$,
(d) the average node degree $d(V', E_{cur} \cup E_{new})$ *is bounded by* $O(d(V, E_{cur}) + \rho \cdot d_{MIN}(S, v'.topics) \cdot \log(|S||v'.topics|))$.

E. Node Join Algorithms with Shadows

Similarly to the *primary-backup* strategy for node leaving, the strategy for node joining can be presented using three steps:
Step I, when v' joins, **build** the *backup set* for v' using nodes in the base TCO;
Step II, repair the overlay "locally" among the *shadow set*, which is $v'.backups \cup \{v'\}$;
Step III, update *backups* and *primaries*.

Fortunately, at the churn round of node joining, there is no need for **update** *backups* and *primaries*, as required for node leaving.

Algorithm 9 Step I: Build Backups for Node Join

buildBackupsOnJoin(v')
Input: joining node v'
Output: $backups$ built for v', $primaries$ updated for $u \in v'.backups$
1: **buildBackupsAt**(v')

Algorithm 10 Step II: Repair TCO for Node Join

repairOverlayOnJoinByShadows($TCO_{cur}(V, T, E_{cur}), v'$)
Input: current overlay TCO_{cur}, and joining node v'
Output: $TCO_{new}(V, T, E_{new})$: new TCO that contains v'
1: $V' \leftarrow V \cup \{v'\}$
2: $S \leftarrow v'.backups \cup \{v'\}$
3: $E_{pot} \leftarrow S \times \{v'\}$
4: $E_{new} \leftarrow$ **constructLowEdges**$(S, v'.topics, E_{cur}, E_{pot})$
5: return $TCO_{new}(V', T, E_{cur} \cup E_{new})$

Alg. 9 **builds** *backups* and corresponding *primaries* for the newly coming node v'.

Alg. 10 **repairs** the TCO for the base overlay TCO_{cur} plus a new joining node v'. The algorithm operates in two phases: First, it computes a *shadow set* S by combining the joining node and its *backup set*, which is built by Alg. 9 at **Step I**, and

then it sets the potential edge set as $\{(v', u)|u \in S\}$. Second, it repairs the TCO by applying Alg. 2 while only considering the shadow nodes including v'.

Both **Step I** and **II** contribute to re-achieving *topic-connectivity* for node joining. However, observe that the runtime complexity of Alg. 9 is of a smaller order of magnitude compared to that of Alg. 10. Lemma 4 is established for the correctness, running time and approximation ratios for our *dynamic* algorithms for node joining.

Lemma 4. *Alg. 10 has the following properties for a node join round:*
(a) the running time is $O(|S|^2|v'.topics|)$,
(b) $G(V', E_{cur} \cup E_{new})$ *is* topic-connected *for the new instance* I',
(c) the maximum node degree $D(V', E_{cur} \cup E_{new})$ *is bounded by* $O(D(V, E_{cur}) + \frac{|S|}{\rho} D_{MIN}(S, v'.topics) \cdot \log(|S||v'.topics|))$,
(d) the average node degree $d(V', E_{cur} \cup E_{new})$ *is bounded by* $O(d(V, E_{cur}) + \rho \cdot d_{MIN}(S, v'.topics) \cdot \log(|S||v'.topics|))$.

F. Analysis of Dynamic Shadow Algorithms under Churn

Alg. 6, 7, 8, 9 and 10 are named *Shadow Dynamic Algorithms* for the Dyn-Low-TCO problem, short for ShadowDynAlg. In this section, we analyze the performance of ShadowDynAlg under a sequence of churn $\Delta = (\delta_1, \delta_2, ..., \delta_k, ...)$.

We assume that each node churn round δ_k only results in an *insignificant* change between instances $I_{k-1}(V_{k-1}, T)$ and $I_k(V_k, T)$. Specifically,

$$D_{MIN}(V_k, T) = O(D_{MIN}(V_{k-1}, T)), \forall \delta_k \quad (3)$$

$$d_{MIN}(V_k, T) = O(d_{MIN}(V_{k-1}, T)), \forall \delta_k \quad (4)$$

Equations (3) and (4) are realistic in pub/sub, but not guaranteed for some corner cases. In [10], we constructed an example such that a single node join results in a significant difference for the minimum average node degree up to $\Theta(|V|)$ times. Similarly, the departure of a node may cause a "earthquake" effect on the node degrees of the optimal TCO. Fortunately, our experimental findings in section VI show that for typical pub/sub workloads, such situations virtually never occur in practice.

Another assumption that we make is: For each churn round δ_k involving a node $v_k{}'$, the *shadow set* S_k can be found such that the minimal possible degrees for instance $I(S_k, v_k{}'.topics)$ formed by the *Shadow set* are not asymptotically larger than the minimal possible degrees for instance $I(V_k, T)$. This is reasonable because we can always set $S_k = V_k$. Formally,

$$D_{MIN}(S_k, v_k{}'.topics) = O(D_{MIN}(V_k, T)) \quad (5)$$

$$d_{MIN}(S_k, v_k{}'.topics) = O(d_{MIN}(V_k, T)) \quad (6)$$

Using Lemma 3 and 4, the following bounds on the maximum and average degrees of the overlay produced by ShadowDynAlg can be derived.

Lemma 5. *Given an instance of the* dynamic *TCO problem* Dyn-Low-TCO$(TCO_0(V_0, T, E_0), \Delta)$, *where* $\Delta = (\delta_1, \delta_2, ..., \delta_k, ...)$ *is composed of an intermixed sequence of single node joins and leaves, suppose* TCO_0 *is constructed by the* static *Alg. 1, and assume that at each churn round* δ_k *equations (3), (4), (5) and (6) are satisfied. Then for any* δ_k, ShadowDynAlg *outputs* $TCO_k(V_k, T, E_k)$ *such that*

$$D(V_k, E_k) = O(\frac{|V_k|}{\rho} D_{MIN}(V_k, T) \cdot \log(|V_k||T|)) \quad (7)$$

$$d(V_k, E_k) = O(\rho \cdot D_{MIN}(V_k, T) \cdot \log(|V_k||T|)) \quad (8)$$

Proof: We prove equation (7) by induction on k. Similar techniques can apply to the proof of equation (8).

Basis case: $k = 0$, TCO_0 is constructed by the *static* Alg. 1; equation (7) immediately holds (Theorem 2 in [7]).

Inductive step: Assume equation (7) is valid when $k = l - 1 (\geq 0)$. We will show that it is also valid for $k = l$.

$$\begin{aligned}
&D(V_l, E_l) \\
&= O(D(V_{l-1}, E_{l-1}) + \frac{|S_l|}{\rho} D_{MIN}(S_l, v_l'.topics) \cdot \log(|S_l||v_l'.topics|)) && \text{// by Lemma 3 and 4} \\
&= O(\frac{|V_{l-1}|}{\rho} D_{MIN}(V_{l-1}, T) \cdot \log(|V_{l-1}||T|)) + O(\frac{|S_l|}{\rho} D_{MIN}(S_l, v_l'.topics) \cdot \log(|S_l||v_l'.topics|)) && \text{// by inductive hypothesis} \\
&= O(\frac{|V_l|}{\rho} D_{MIN}(V_l, T) \cdot \log(|V_l||T|)) + O(\frac{|V_l|}{\rho} D_{MIN}(V_l, T) \cdot \log(|V_l||T|)) && \text{// by equation (3) and (5)} \\
&= O(\frac{|V_l|}{\rho} D_{MIN}(V_l, T) \cdot \log(|V_l||T|))
\end{aligned}$$

■

Therefore, under realistic and reasonable assumptions, ShadowDynAlg will *dynamically* maintain a TCO whose maximum and average node degrees are asymptotically identical to those of the TCO output by *static* Alg. 1.

G. Building Shadows

We now consider the problem of the initial construction of backup sets. It is easy to show that a *backup set* that satisfies equations (1) and (2) is equivalent to a *set cover* where v''s topics is the universe of ground elements that are supposed to be covered by the topic set of *backup nodes*. Classical algorithms for the *minimum weight set cover* problem are capable of producing a small but *feasible backup set*. With regard to runtime cost, small cardinality is desired for *backup set*, because the number of *backups* directly impacts the size of the *shadow set*, and thus the time complexity of ShadowDynAlg (Lemma 3 and 4).

On the other hand, equations (5) and (6) express that a sufficiently large *backup set* is preferred for ensuring the quality of the output TCO: a larger *backup set* means a large *shadow set* S, and therefore a higher probability for the instance $(S, v'.topics)$ to approximate $I'(V', T)$ with regard to maximum and average node degrees in the TCO.

We consider the notion of *coverage factor* for tuning the size of the *backup set*, and seeking the balance between time complexity of overlay maintenance and the quality of the output TCO.

Given a Low-TCO instance $I(V, T)$ and a node subset $U \subseteq V$, for topic $t \in T$ that is subscribed by some node in U, denote $t.nodes|_U = \{u | u \in U \wedge t \in u.topics\}$. The coverage factor for U, denoted as $\lambda|_U$(or λ), is the minimum size of the *subscriber node set* within U for any topic $t \in T$:

$$\lambda|_U = \min_{t \in T} \left| t.nodes|_U \right|$$

The *coverage factor* is a positive integer ($\lambda \geq 1$) such that each topic of interest is covered at least λ times.

The coverage factor selection exhibits the tradeoff between the running time and node degrees: On one side, $\lambda = 1$ minimizes the size of *backup set*, and running time, but leads to a severe influence on the node degrees, especially since equations (5) and (6) are not likely to hold with a small *shadow set*. On the other side of the spectrum, if we choose the *coverage factor* to be a large value such that all nodes of V' has to be included in the *backup set*, then the behavior of ShadowDynAlg becomes identical to that of NaiveDynAlg. According to our experiments in subsection VI-A, an increase in λ beyond 3 only marginally improves the node degrees of the TCO under churn. The *backup set* for $\lambda = 3$ is significantly smaller than the complete node set itself so that we choose 3 as the default value for λ.

Alg. 11 and 12 provide two different approaches to **build** the *backup set* for a particular node v with specified *coverage factor* λ. To ensure the *coverage factor* of the *backup set*, basic approximation algorithms for the *minimum weight set cover* are executed λ times, and the generated *backup set* is the union of λ disjoint *feasible set covers* that approximate the minimum.

Minimum weight set cover is known to be NP-hard, but approximation algorithms for this problem are well studied. Basically, there are two efficient approximation algorithms for *minimum weight set cover*: 1) greedy algorithm [24], and 2) primal-dual algorithm [25]. These two approximations serve as foundations to build a λ-*backup-set* in Alg. 11 and 12 respectively.

Algorithm 11 Greedy Algorithm to Build Backups

buildBackupsGreedilyAt(v, λ)
Input: node v, coverage factor λ
Output: $backups$ built for v, and $primaries$ updated for $w \in v.backups$

$v.backups \leftarrow \emptyset$
repeat
  $T_{rest} \leftarrow v.topics$
  $V_{can} \leftarrow V \backslash (\{v\} \cup v.backups)$
  while $T_{rest} \neq \emptyset \wedge V_{can} \neq \emptyset$ **do**
    $V_{can} \leftarrow \{u | u \in V_{can} \wedge u.topics \cap T_{rest} \neq \emptyset\}$
    for all node $u \in V_{can}$ **do**
      $weight(u) = |u.neighbors| + |u.primaries|$
    node $w \leftarrow \text{argmin}_{u \in V_{can}} \frac{weight(u)}{|u.topics \cap T_{rest}|}$
    $v.backups \leftarrow v.backups \cup \{w\}$
    $w.primaries \leftarrow w.primaries \cup \{v\}$
    $T_{rest} \leftarrow T_{rest} - w.topics$
  $\lambda \leftarrow \lambda - 1$
until $\lambda = 0$

Algorithm 12 Primal-Dual Algorithm to Build Backups

buildBackupsByPDAt(v, λ)
Input: node v, coverage factor λ
Output: $backups$ built for v, and $primaries$ updated for $w \in v.backups$
1: $v.backups \leftarrow \emptyset$,
2: **repeat**
3: $T_{rest} \leftarrow v.topics$
4: **for all** node $u \in V - (\{v\} \cup v.backups)$ **do**
5: $weight(u) = |u.neighbors| + |u.primaries|$
6: **while** $T_{rest} \neq \emptyset$ **do**
7: topic $t \leftarrow$ randomly pick an uncovered topic from T_{rest}
8: $V_{can} \leftarrow \{u : t \in u.topics \wedge weight(u) > 0\}$
9: node $w \leftarrow \text{argmin}_{u \in V_{can}} weight(u)$
10: $v.backups \leftarrow v.backups \cup \{w\}$
11: $w.primaries \leftarrow w.primaries \cup \{v\}$
12: $T_{rest} \leftarrow T_{rest} - w.topics$
13: **for all** node $u \in V_{can}$ **do**
14: $weight(u) \leftarrow weight(u) - weight(w)$
15: $\lambda \leftarrow \lambda - 1$
16: **until** $\lambda = 0$

Alg. 11 applies the *greedy set cover* algorithm [24]. It follows a commonly intuitive criterion in choosing node from V', towards construction of a feasible *backup set*: *always choose the next node w which provides the minimum average weight over uncovered topics that would be covered by w* (line 9 of Alg. 11). This *greedy* algorithm is proven to achieve a log approximation ratio compared to the optimum solution for the *minimum weight set cover* problem. Along with its polynomial time complexity, *greedy* is usually the method of choice for many practical applications. However, this *greedy* algorithm does not fit into our *dynamic* TCO settings, in which a number of *set cover* problems for each *backup set* need to be taken care of together. *Greediness* tends to aggregate *backup* burdens to small portion of bulk nodes that subscribe to a large number of topics. A small number of bulk nodes would serve as *backups* for a large number of *primaries*; while the majority of lightweight nodes bear very low *primary degree*. Fairness is lost to a large extent in this case, and it is against our target of load balancing; the impact of accumulated sub-optimality would become increasingly severe as more and more churn occurs.

Alg. 12 uses another another approximation algorithm for *minimum weight set cover*, which is thought of as an instance of *primal-dual* method. The algorithm proceeds in an iterative manner: *each time randomly pick an uncovered topic t, and then choose node w as a* backup *which has the minimum weight* (lines 7-9 of Alg. 12). The *primal-dual* algorithm yields a f-approximate solution for *minimum weight 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. However discouraging this approximation ratio may seem, in practice the *primal-dual* algorithm still achieves high quality solutions on many instances of *minimum weight set cover*. Moreover, the *primal-dual* algorithm integrates randomness into greediness, and effectively reduces the "monopoly" of bulk subscribers, which renders the *greedy set cover* algorithm inferior in our *dynamic* TCO environment (see evaluation in subsection VI-A).

We decide to leverage on the *primal-dual* algorithm towards **building** and **updating** the *backup set* in **Step I** and **III**. An empirical comparison is shown in Section VI-A.

VI. Evaluation

We implemented all algorithms in Java and evaluated them under a variety of experimental settings. We use Low-ODA as a baseline because it is the only known polynomial time algorithm that achieves a sub-linear approximation ratio on both average and maximum node degrees. Parameter ρ is chosen to be 3 as suggested in [7].

In our experiments, the base instance $I_0(V_0, T)$ was initialized with $|V_0|$=2 000, $|T|$=200 and $|v.topics| \in [10, 90]$, where the *subscription size* of each node follows a power low. Each topic $t \in T$ is associated with a node with probability p_t, $\sum_{t \in T} p_t$=1 (i.e., each node subscribes to t with a probability p_t). The value of p_t is distributed according to either a uniform, a Zipf (with α=2.0), or an exponential distribution. According to [18], these distributions are representative of actual workloads used in industrial pub/sub systems today. The Zipf distribution is chosen because Liu *et al.* [3] show it faithfully describes the feed popularity distribution in RSS feeds (a pub/sub-like application scenario) and the exponential distribution is used because it was convincingly applied in stock-market monitoring scenarios for modelling the popularity of stock in the NYSE [26].

All dynamic behavior of the algorithms is evaluated over 1 000 rounds of churn. For each churn round, a join or leave is generated with probability 0.5. The subscriptions of joining nodes follow the same distribution as those of the nodes in $|V_0|$.

A. Building Backups Greedily vs. by Primal-dual

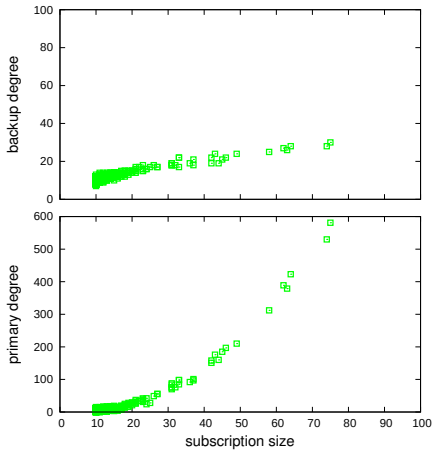

Figure 2. Building backups greedily

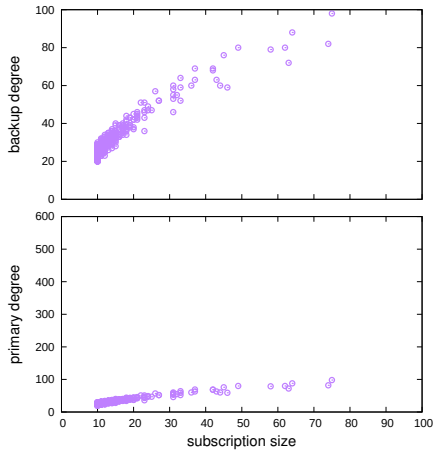

Figure 3. Building backups by primal-dual

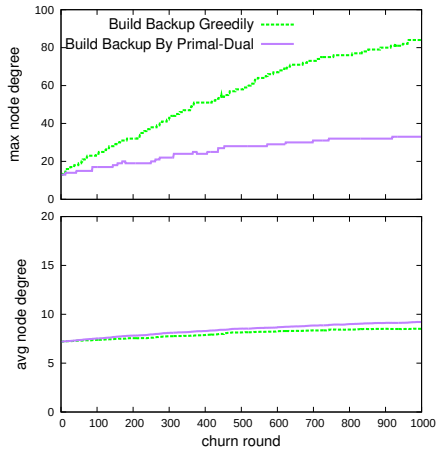

Figure 4. TCO properties under churn

We first compare different algorithms for building *primary-backups*. Given the same instance $I_0(V_0, T)$, Alg. 11 and Alg. 12 are used to **build** *backups* with λ=3 at **Step I**. Fig. 2 and Fig. 3 show the output of both *primary-backup* schemes under a uniform distribution as a function of the *subscription size*, respectively. As shown in the figures, both *greedy* and *primal-dual* algorithms produce small-sized *backup sets* compared to the complete node set V_0: 1.1% of $|V_0|$ for the *greedy* algorithm and 2.8% of $|V_0|$ for *primal-dual*, averaged across all nodes. In general, the *backup degree* is linearly proportional to the *subscription size* for both *greedy* and *primal-dual* algorithms. However, the distribution of *primaries* differs considerably between both algorithms. The *primaries* produced by the *greedy* algorithm are skewed and follow an exponential shape: 42.9% of all nodes have less than 5 *primaries*, while the highest *primary degree* of a single bulk subscriber peaks at 581. At the same time, the distribution of backups produced by the *primal-dual* algorithm is well-balanced, and the shape of the dependency is linear: the lowest *primary degree* is 21 and the highest is 59. These results confirm our observations in Section V-G about the side effect of *greediness* on the *primary* assignment and the fairness introduced by randomness in the *primal-dual* scheme.

We also compare both backup construction algorithms in terms of the evolution of overlay properties under churn. As shown in Fig. 4, both maximum and average node degrees increase as the TCO instance evolves with node churn. However, overlay quality for *Shadow Dynamic* utilizing the *greedy* algorithm degrades noticeably, i.e., at δ_{1000}, the maximum node degree becomes 84. On the other hand, when *backups* are built by *primal-dual*, both maximum and average degrees keep a steadily low growth rate.

Results presented in Fig. 2, 3 and 4 substantiate our choice of *primal-dual* for building *backups*.

B. Impact of Coverage Factor

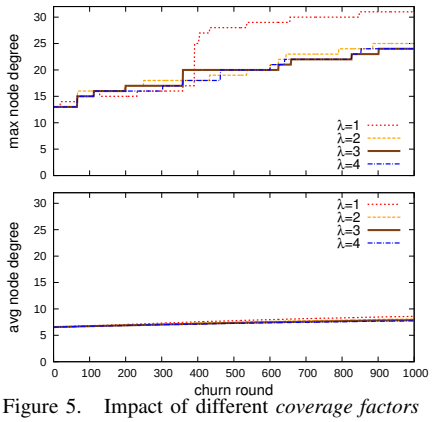

Figure 5. Impact of different *coverage factors*

Next, we explore the impact of λ on the output and performance of the *Shadow Dynamic Algorithm*. Given an initial input $I_0(V_0, T)$ where $|V_0| = 2000$ and $|T| = 200$, and a sequence of $1\,000$ rounds of node joins and leaves, ShadowDynAlg is

evaluated for four different values of the coverage factor ($\lambda = 1, 2, 3, 4$). Fig. 5 shows that, as λ increases, D_{Shadow} decreases, and its growth rate with respect to node churn decreases. The differences in maximum node degrees and their growth rates also *decrease* with successive coverage factors. For example, under δ_{1000}, $D_{\mathsf{Shadow}}|_{\lambda=1} = 31$, $D_{\mathsf{Shadow}}|_{\lambda=2} = 25$, and $D_{\mathsf{Shadow}}|_{\lambda=3} = D_{\mathsf{Shadow}}|_{\lambda=4} = 24$. When $\lambda \geq 3$, the difference is insignificant. This experiment confirms the validity of choosing a relatively small coverage factor as noted in Section V-G.

C. Performance Under Sequences of Node Churn

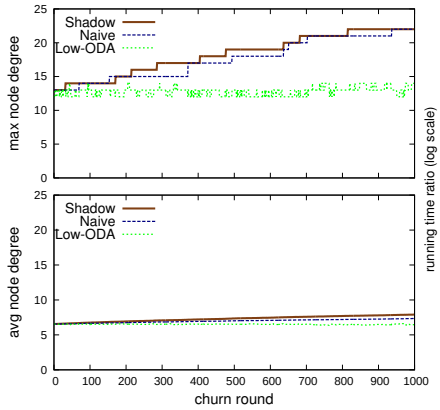

Figure 6. Overlay evolution under churn

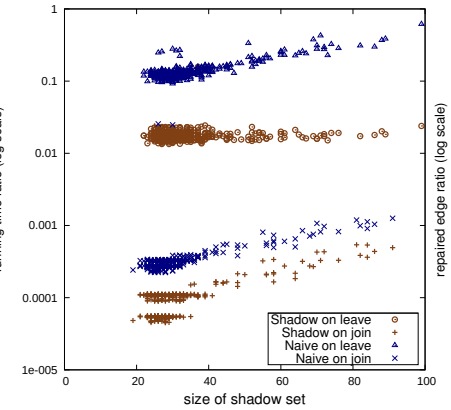

Figure 7. Comparison of running time ratios

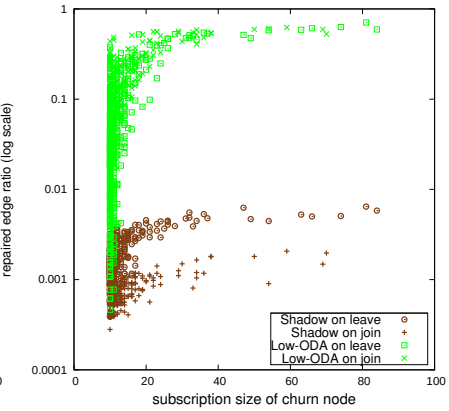

Figure 8. Comparison of repair ratios

We now demonstrate the scalability, efficiency and robustness of our ShadowDynAlg under node churn.

Fig. 6 depicts a comparison between *Shadow*, *Naive* and Low-ODA algorithms as the instances evolve with the random churn sequence Δ. The figure shows that ShadowDynAlg and NaiveDynAlg output similar TCOs with regard to maximum and average node degrees, which are slightly higher than D_{Low} and d_{Low}. However, the differences are insignificant: $\overline{D_{\mathsf{Shadow}} - D_{\mathsf{Low}}} \leq 5.5$, and $\overline{d_{\mathsf{Shadow}} - d_{\mathsf{Low}}} \leq 0.798$ on average over the entire sequence of churns.

Furthermore, the average node degrees of our *dynamic* algorithms (both *Shadow* and *Naive*) do not degrade significantly under churn; the maximum node degree increases *slowly* like a step function. The figure shows that, from churn δ_1 to δ_{1000}, $D_{\mathsf{Shadow}} (= D_{\mathsf{Naive}})$ increases from 13 to 22, a rate of 0.9%.

Fig. 7 depicts the running time ratio of our ShadowDynAlg against Low-ODA as a function of the size of *shadow set* for different joins and leaves in the churn sequence, i.e., each dot in the figure reflects the *shadow size* at one of the churn rounds and the time ratio for that round. We plot the NaiveDynAlg against Low-ODA and set the *shadow size* of the NaiveDynAlg to the same value as for ShadowDynAlg at the same churn round, since the NaiveDynAlg does not have a *shadow set*.

As shown in Fig. 7, ShadowDynAlg is considerably faster than NaiveDynAlg both for node joins and leaves. On average, the running time ratio against Low-ODA is around 0.014% for *Shadow* (and 0.043% for *Naive*) under joins and 1.78% for *Shadow* (and 14% for *Naive*) under leaves. The runtime cost for the NaiveDynAlg can be as much as 61.6% of that for Low-ODA. More specifically, it takes Low-ODA 18.76 seconds on average to compute a new TCO at each churn round; NaiveDynAlg takes 7.93 milliseconds for a join round, and 2575 milliseconds for a leave round. Yet, ShadowDynAlg only takes 2.69 milliseconds for a join round, and 333 milliseconds for a leave round. This can be explained by different *shadow sets*, and accordingly, different potential edge sets for the *Shadow* and *Naive Dynamic Algorithms*. The *shadow set* S is much smaller than the complete node set V'; the ratio is 1.59% on average and $\leqslant 4.97\%$ in the worst case in our experiments.

Finally, we present the *repair ratio* as a metric to show the non-intrusiveness of our algorithm in *dynamic* TCO settings. We refer to the number of edges that an algorithm alters (either adds or deletes) in order to evolve TCO_{k-1} to TCO_k at churn round δ_k as the *repair edge number*. The *repair ratio* is the ratio of *repair edge number* to the number of edges in the base overlay, i.e. $|E_{k-1}|$. Note that edges shared by both TCO_{k-1} and TCO_k do not contribute to the values of *repair edge number* and *repair ratio*. Fig. 8 compares the *repair ratios* of ShadowDynAlg and Low-ODA as a function of the *subscription size* of churn node for different churn rounds in the given Δ sequence. The *repair ratio* of ShadowDynAlg is significantly lower than that of the *static* Low-ODA: The average *repair ratio* of Low-ODA is 9.4%, 11.3% for joins and 7.5% for leaves while the ratio of ShadowDynAlg is 0.068% for joins and 0.17% for leaves. In other words, after computing the TCO, Low-ODA has to perform 610 edge additions and deletions on average, to migrate from the old TCO_{k-1} to the new TCO_k. The average number of required edge changes for ShadowDynAlg is 4.83 for a join round, and 12.24 for a leave round.

VII. Conclusions

We present a fully *dynamic* algorithm, ShadowDynAlg, for the new problem of Dyn-Low-TCO. ShadowDynAlg dynamically maintains an approximate solution such that, under realistic assumptions about typical pub/sub workloads, the approximation ratio for both maximum and average degree is asymptotically identical to that of state-of-the-art *static* algorithms. We have presented an extensive experimental evaluation of the algorithms. Our proposed ShadowDynAlg is demonstrated to efficiently and robustly maintain a TCO with low degrees.

References

[1] J. Reumann, "Pub/Sub at Google," lecture & Personal Communications at EuroSys & CANOE Summer School, Oslo, Norway, Aug'09.
[2] "Tibco rendezvous," http:\www.tibco.com.
[3] H. Liu, V. Ramasubramanian, and E. G. Sirer, "Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews," in *IMC'05*.
[4] 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.
[5] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Constructing scalable overlays for pub-sub with many topics: Problems, algorithms, and evaluation," in *PODC'07*.
[6] M. Onus and A. W. Richa, "Minimum maximum degree publish-subscribe overlay network design," in *INFOCOM'09*.
[7] ——, "Parameterized maximum and average degree approximation in topic-based publish-subscribe overlay network design," in *ICDCS'10*.
[8] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann, "Self-organizing broker topologies for publish/subscribe systems," in *SAC'07*.
[9] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito, "Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA," *Comput. J.*, vol. 50, no. 4, 2007.
[10] C. Chen, H.-A. Jacobsen, and R. Vitenberg, "Divide and conquer algorithms for publish/subscribe overlay design," in *ICDCS'10*.
[11] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in *PODC*, 2002.
[12] E. De Santis, F. Grandoni, and A. Panconesi, "Fast low degree connectivity of ad-hoc networks via percolation," in *ESA'07*.
[13] L. C. Lau, J. S. Naor, M. R. Salavatipour, and M. Singh, "Survivable network design with degree or order constraints," in *Proc. ACM STOC'07*.
[14] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed, "Magnet: practical subscription clustering for internet-scale publish/subscribe," in *DEBS'10*.
[15] C. Chen, R. Vitenberg, and H.-A. Jacobsen, "Scaling construction of low fan-out overlays for topic-based publish/subscribe systems," in *ICDCS'11*.
[16] M. Matos, A. Nunes, R. Oliveira, and J. Pereira, "Stan: exploiting shared interests without disclosing them in gossip-based publish/subscribe," in *IPTPS'10*.
[17] 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*.
[18] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication," in *DEBS'07*.
[19] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *JSAC*, 2002.
[20] E. Baehni, P. Eugster, and R. Guerraoui, "Data-aware multicast," in *DSN'04*.
[21] R. Chand and P. Felber, "Semantic peer-to-peer overlays for publish/subscribe networks," in *EUROPAR'05*.
[22] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Constructing scalable overlays for pub-sub with many topics: The greedy join-leave algorithm," US Patent Application US 2010/0 027 442 A1.
[23] C. Chen, R. Vitenberg, and H.-A. Jacobsen, "A generalized algorithm for publish/subscribe overlay design and its fast implementation," U. of Toronto & U. of Oslo, Tech. Rep., 2011, http://msrg.org/papers/TRCVJ-GenODA-2011.
[24] V. Chvtal, "A greedy heuristic for the set-covering problem," *Mathematics of Operations Research*, 1979.
[25] D. S. Hochbaum, "Approximation algorithms for the set covering and vertex cover problems," *SIAM Journal on Computing*, 1982.
[26] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky, "Hierarchical clustering of message flows in a multicast data dissemination system," in *IASTED PDCS*, 2005.