BeaConvey: Co-Design of Overlay and Routing for Topic-based Publish/Subscribe on Small-World Networks

Chen Chen
Middleware System Research Group
University of Toronto
chenchen@eecg.toronto.edu

Yoav Tock
IBM Research - Haifa
tock@il.ibm.com

Sarunas Girdzijauskas
Royal Institute of Technology (KTH), Sweden
sarunasg@kth.se

Abstract

Distributed pub/sub must make principal design choices with regards to overlay topologies and routing protocols. It is challenging to tackle both aspects together, and most existing work merely considers one. We argue the necessity to address both problems simultaneously, since only the right combination of the two can deliver an efficient internet-scale pub/sub. Traditional design space spans from structured data-oblivious overlays employing greedy routing strategies all the way to unstructured data-driven overlays using naive broadcast-based routing. The two ends of the spectra come with unacceptable prices: the former often exerts considerable overhead on each node for forwarding irrelevant messages, while the latter is difficult to scale due to prohibitive latencies stemming from unbounded node degrees and network diameters.

To achieve the best of both worlds, we propose BeaConvey, a distributed pub/sub system for federated environments. First, we define the *small-world and interest-close overlay* (SWICO) that embraces both small-world properties and pub/sub semantics. To construct a SWCIO, we devise a greedy heuristic to assign small-world identifiers and fingers in a centralized manner. Second, we develop a family of peer-to-peer pub/sub routing protocols that leverages such SWICOs.

Empirical evaluation shows that BeaConvey achieves substantial improvement in routing overhead and propagation delays. For instance, the routing overhead of BeaConvey is only 20% to 40% of the state of the art. This acceleration is consistent across a variety of pub/sub workloads, and BeaConvey obtains such adaptability by optimizing both overlay and routing, which complement each other in different situations. Under one Facebook workload with a skewed distribution, 78% of the improvement is accredited to a better overlay. Under another non-skewed workload, more advanced routing contributes 95% of cost reduction.

CCS Concepts

• **Information systems** → **Enterprise applications**; **Data centers**; • **General and reference** → *Design*;

Keywords

pub/sub, overlay, routing, topic-connected overlay, small-world

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DEBS '18, June 25–29, 2018, Hamilton, New Zealand
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5782-1/18/06…$15.00
https://doi.org/10.1145/3210284.3210287

ACM Reference Format:
Chen Chen, Yoav Tock, and Sarunas Girdzijauskas. 2018. BeaConvey: Co-Design of Overlay and Routing for Topic-based Publish/Subscribe on Small-World Networks. In *DEBS '18: The 12th ACM International Conference on Distributed and Event-based Systems, June 25–29, 2018, Hamilton, New Zealand.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3210284.3210287

1 Introduction

Publish/Subscribe (pub/sub) systems constitute an attractive choice as communication paradigm and messaging substrate for building large-scale distributed systems. This work concentrates on the *topic-based pub/sub* model: the system disseminates messages on abstract channels called topics, publishers associate each publication message with one or more specific topics, and subscribers register their interests in a subset of all topics. Many real-world applications adopt topic-based pub/sub for message dissemination, such as Internet of things [4], big data platforms [1], application integration across data centers [13, 32], RSS feeding [24], etc.

A distributed pub/sub system often organizes nodes (e.g., brokers, servers or routers) in a federated or peer-to-peer manner as an overlay at the application or network layer. Once an overlay network is constructed, the pub/sub system relies on some routing protocol to build message dissemination paths for delivering publications to all subscriber nodes. Typically, a pub/sub routing protocol defines the forwarding function at each node, which determines the set of next-hops for an incoming publication message. Overlay topologies and routing protocols are closely related – both impact the performance and scalability of the pub/sub system.

It is a fundamental challenge to seamlessly glue both overlay and routing together in a pub/sub system design. A significant body of research has been centered around either routing or overlay alone. In many earlier pub/sub systems [6, 23], routing protocols took all the heavy lifting, assuming the most generic overlays, such as trees or full-meshes. In a typical pub/sub routing protocol where the underlying overlay is simply a tree, each node v needs to maintain a forwarding table, i.e., a map between v's neighbors and predicates. Each predicate p_n corresponds to the union of the subscriptions of all downstream nodes reachable through node n, one of v's neighbors. To support pub/sub routing, many nodes have to maintain a global view of all subscriptions on all nodes in the system. Consequently, such pub/sub systems suffer from large forwarding tables, excessively high matching complexity, reliance on selective message flooding, expensive routing computations, etc. Recently, many pub/sub systems try to migrate part of the complexity to overlay design [7, 12, 29, 35, 41]. A well-constructed overlay could potentially simplify the pub/sub routing protocols and improve the efficiency of message dissemination.

We propose BeaConvey, a distributed pub/sub system deployed in one data center. We carefully build the overlay in a centralized manner and develop peer-to-peer pub/sub routing protocols. As compared to canonical pub/sub routing based on trees or full-mesh, BeaConvey stems from a grander design space and achieves better performance by tackling more challenges; for example, each BeaConvey node $v \in V$ only maintains a partial knowledge of size $O(\log |V|)$, and the path length of any message is $O(\log |V|)$.

1.1 Overlay topologies

We build the *small-world and interest-close overlay* (SWICO). First, we exploit small-world networks [14, 20, 25, 36], especially because pub/sub routing greatly benefits from navigability of these structured overlays. A small-world network assigns each node a random and unique *identifier* (ID), a *coordinate* in the ID space, and all nodes are in agreement of quantitative *distances* between any two nodes. Nodes link to each other with probability inversely proportional to their distances, and the resulting network becomes efficiently searchable even if there are only bounded number of such links per node, i.e., each node can locate any other node by a limited number of hops with only local and partial knowledge. Second, to further optimize pub/sub message dissemination, the overlay should maximize *interest closeness*, meaning that nodes with similar interests stay topologically close to each other. One way to concretize interest closeness is to enforce a *topic-connected overlay* (TCO) [11, 12]. In a TCO, each topic t induces a connected sub-overlay among all nodes interested in t. TCO can thereby support the transmission of publications on each topic to all subscribers without using non-interested nodes as intermediate relays.

Unfortunately, small-world networks and TCOs are at odds with each other. It is NP-hard to construct a TCO with a fixed node degree [11, 27], while a small-world network strictly restricts each node to possess a bounded number of fingers in each small-world phase. We decide to build a *partial* TCO, a relaxation of the TCO requirement, while maintaining the small-world properties. To quantify how close a partial TCO approximates a complete one, we define the *TCO support ratio* [9], which grows monotonically from 0 to 1 as the overlay expands from none to a TCO. We study the problem of SWICO design on maximization of TCO support ratio while forming a small-world network. The global knowledge of pub/sub subscriptions in the centralized master endows us with the possibility to fully optimize SWICO design for pub/sub routing.

We construct a SWICO in an one-dimensional space, which, informally speaking, models a small-world network as a ring augmented with structured edges [17, 25, 31, 36]. The edge set consists of two parts: (1) *short*-range links that jointly constitute the ring and determine node IDs, and (2) *long*-range links, i.e., the remaining edges that are not on the ring. Once the ring is set (i.e., short-range links are provided), long-range links permit a certain degree of randomness and flexibility. Hence, we construct a SWICO with two steps: Step I selects short-range links and assign node IDs uniformly at random across the entire ID space along the ring; Step II chooses long-range links. In both steps, our algorithms share the same essence of greediness to maximize TCO support: we add edges one by one, always selecting an edge with the highest contribution towards TCO under the constraint of small-world networks.

The overlay design relies on a centralized master with global knowledge, which is conceptually similar to the role of root nodes in Google clouds [18] and name nodes in Hadoop [3]. Like these existing systems, this centralized design is feasible within data center environments, because our master is only responsible for a limited set of centralized operations (e.g., overlay design) and does not reside in the critical data flow paths for pub/sub messaging. Besides, the global knowledge of pub/sub subscriptions can easily fit into the main memory of one modern machine.

1.2 Routing protocols

We devise peer-to-peer pub/sub routing protocols atop SWICO. The nice properties of small-world networks allow us to disseminate messages in a *divide-and-conquer* manner: upon message arrival, each node (1) divides the original dissemination range into a number of sub-ranges based on a set of carefully selected next-hops and (2) conquers each sub-range independently and recursively. The crux of our pub/sub routing lies in the selection of next-hops for each incoming message, and we strike the balance between routing overhead and message latency by leveraging the pub/sub semantic knowledge (i.e., subscription interests of each node).

We devise three implementations of selecting next-hops: `Po`, `Pal`, and `Pie`. The first function `Po` strives to minimize routing overhead, and its next-hop array combines two parts: (1) the nearest subscriber of the given topic t along the ring and (2) all small-world fingers of node v that subscribe to t. However, `Po` solely optimizes routing overhead and may perform poorly on message latency. `Po`'s drawback motivates us to develop the second function `Pal`, which extends the next-hop of `Po` by *always* securing a *pivot*, a node from the second half of the targeted range of the message. `Pal` guarantees to at least halve the targeted range at each recursive call, and hence the maximum path length is bounded by a logarithmic factor. Unfortunately, `Pal` may incur significant extra routing overhead, as pivots may not be interested in t. To combine the strengths from both `Po` and `Pal`, we develop `Pie`, which selectively adds a pivot only if the pivot is a small-world finger of the node. By doing this, `Pie` achieves a sound balance between routing overhead and propagation delay.

1.3 Churn handling

BeaConvey targets data center application scenarios with moderate churn [2, 4, 32, 34]: the intervals between successive churn events are in the order of hours or tens of hours. Two types of churn events are in consideration: (1) node churn (join, leave or fail) and (2) subscription churn (subscription or unsubscription).

BeaConvey assures correctness under both types of churn events by relying on the large body of work that addresses node churn in distributed and dynamic environments, such as ring and finger maintenance [17, 25, 33, 36]. Still, sub-optimality may accumulate due to continuous node or subscription churn. Fortunately, this is not a serious issue for many real-world applications that reside upon BeaConvey, since the overlay is reconstructed periodically (e.g., daily or weekly) in the centralized master, while the churn rates of both types are sufficiently low and do not have a noticeable impact between the periodic recalculations.

We show analytically and empirically that it is important to combine both overlay topologies and routing protocols in the design of a pub/sub system and that BeaConvey achieves substantial balanced benefits in routing overhead and message latencies. Under skewed distributions, overlay prevails over routing in making BeaConvey scalable. For instance, in a Facebook workload, the amount of pure forwarding messages in BeaConvey is only 0.266 the cost of the traditional rendezvous routing: about 78% of this improvement stems from a better overlay, and the remaining 22% is accredited to the more advanced routing. Under synthetic non-skewed distributions, routing becomes increasingly dominant for performance acceleration: under an uniform distribution, BeaConvey yields only 0.379 the routing overhead of the rendezvous routing: 94.9% comes from routing, and 5.1% belongs to overlay.

2 Related Work

To improve the performance and scalability of distributed pub/sub systems, two directions have crystallized in the literature: (1) the design and implementation of routing protocols such that publications and subscriptions are distributed in a most efficient way across the overlay network (see [23, 37]) and (2) the construction of the overlay topology such that network traffic is minimized (e.g., [6, 11, 16, 19, 27]).

Most pub/sub implementations concentrate on one aspect only. Some pub/sub systems just employ the most naive overlays (e.g. a tree or full-mesh) and push all efforts on the routing protocols [6, 23]. These pub/sub routing protocols are difficult to scale in nature, because they inevitably rely on sophisticated matching engines and large forwarding tables. Many pub/sub systems strive to migrate part of the complexity from routing protocols to overlay design [7, 12, 29, 35, 41]. We can generally classify current pub/sub overlays into two major categories: (1) structured and (2) unstructured. Structured overlays organize all nodes in an ID space where everyone has a measurable sense about relative locations and distances with each other, while unstructured ones do not have it.

Structured overlays have been widely used in various distributed systems. In particular, the structures of small-world networks (originated in [20, 21]) have inspired several popular DHT designs, e.g., Chord [36] and Symphony [25]. Small-world networks also provide solid groundwork for our BeaConvey design and many other pub/sub systems [7, 41]. Still, pub/sub overlays are fundamentally different from these canonical distributed networks. For example, a DHT [17, 20, 25, 33, 36] maps IDs (as keys) to nodes – this mapping is self-dependent but determines the overlay topology and routing scheme; DHTs are ID-centric and thus inappropriate for organizing nodes that are semantically related, while pub/sub also needs to accommodate additional semantic information, e.g., topic interests at each node. Hence, pub/sub overlay design poses unique challenges for distributed systems, such as (a) construction of a semantic overlay; (b) support of routing protocols for subscription placement, interest matching, message dissemination, etc.; and (c) scalability with the number of topics (i.e., diverse interests), subscription sizes, and the volume of publications.

Scribe [7] and Bayeux [41] adopt the structured small-world networks and devise rendezvous routing on top of typical DHTs [33, 40]. However, they suffer from excessive amount of routing overhead at each node for forwarding irrelevant messages that do not match the interests of the node. This is not surprising, since classic DHTs do not exploit semantic information about topic interests.

Vitis [29, 30] targets at eliminating pure forwarders in Scribe-like rendezvous routing; it extends the small-world network by adding a bounded number of *friend* connections, which lean towards nodes with similar interests. Some works [15, 16] manipulate the ID assignment of DHT so that nodes with similar interests are close to each other in the ID space. A detrimental consequence of this manipulation is that the nice properties of consistent hashing are often broken. Those techniques are orthogonal to our design and may be used to enhance BeaConvey in future.

Unstructured overlays are also appealing candidates for pub/sub overlay design. The TCO property is explicitly enforced in [5, 12, 28, 35] and implicitly manifest in [16, 29, 30], because TCO effectively reduces unnecessary intermediate overlay hops for message delivery. However, it is unrealistic to keep node degrees or diameters of the TCO bounded, even if we can apply the state-of-the-art centralized algorithms [11, 12, 27].

Among all existing pub/sub systems, Scribe [7] is closest to BeaConvey – both develop pub/sub routing protocols over small-world networks and do not rely on extra edges.

In [8], we construct a a *partial* TCO that attains the small-world properties, which gains over 30% of reduction in the costs of both routing overhead and message latency. This initial exploration provides us a solid guidance for BeaConvey.

3 Overlay: small world and interest closeness

3.1 Small-world networks

We abstract a distributed topic-based pub/sub system as an instance (V, T, I), where V is a node set, T is a topic set, and I is the interest function such that $I : V \times T \to \{0, 1\}$. Node $v \in V$ is interested in topic $t \in T$ iff $I(v, t) = 1$. We also say node v subscribes to topic t.

Our pub/sub system design relies on a one-dimensional small-world network model [25, 33, 36]. We arrange all nodes along a ring and equip each node with a handful of small-world fingers, whose distributions roughly admit inverse proportionality to the small-world distances.

Each node $v \in V$ has an unique ID from a one-dimensional cyclic ID space, which we can draw uniformly at random or construct specifically (see §3.2). For clarity and conciseness, we assume that each node ID has $\log |V|$ bits, and consequently exactly one node is present for every identifier in the space. Practical systems typically use $\log N$ bit IDs where $N \gg |V|$, so nodes do not fully populate the entire ID space, which is essential for churn handling, e.g., node joins and departures. This assumption facilitates the presentation while not affecting the correctness of our system design [17], since dynamic churn is not the focus of this work. As we noted before, our BeaConvey design inherits churn resilience from small-world networks, which has been extensively studied [17, 25, 33, 36].

The small-world distance from node v to node w, which we denote by $\texttt{swDistance}(v, w)$, is the clockwise numeric distance from v to w on the circle[1]. We say that node w (or edge $e = (v, w)$) is in the i-th small-world phase of node v, if $\texttt{swDistance}(v, w) \in \left[2^i, 2^{(i+1)}\right)$, i.e., $w \in \left[v + 2^i, v + 2^{(i+1)}\right)$; we also denote the small-world phase as $v.\texttt{swPhase}(w) = v.\texttt{swPhase}(e(v, w)) = i$.

Property 1. *In a network of the node set V, each node $v \in V$ maintains $k = \Theta(\log |V|)$ small-world fingers, and its i-th finger, $v.swFinger[i]$ where $0 \le i < k$, points to a node in the i-th small-world phase of v, i.e., $v.swFinger[i] \in \left[v + 2^i, v + 2^{(i+1)}\right)$.*

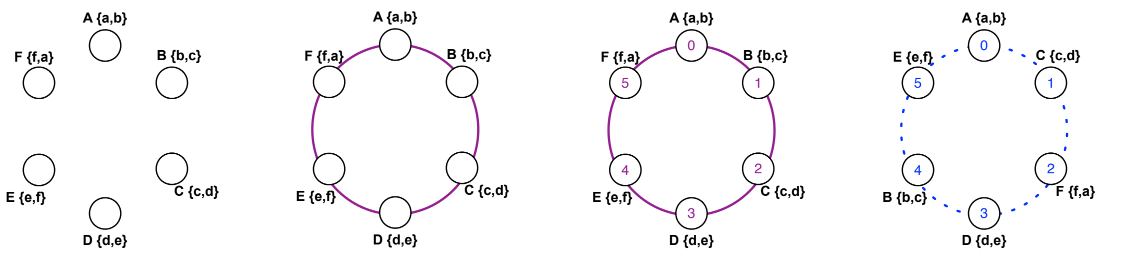
For a node v, we refer to $v.swFinger[0]$ as the *short*-range finger, i.e., v's immediate neighbour clockwise, while the *long*-range fingers are $v.swFinger[i]$ where $1 \le i < k$.

Many DHTs attain this Property 1 [25, 33, 36]. In Chord [36], for instance, node $v \in V$ keeps exactly $k = \log |V|$ fingers, where $v.swFinger[i] = v + 2^i, 0 \le i < k$. As [17] points out: although Chord defines specific small-world fingers for each node, this rigidity is not critical, and small-world networks allow flexibility for finger selection; specifically, greedy routing between any two nodes is still $O(\log |V|)$ hops, even if node $v \in V$ picks $v.swFinger[i]$ as any node in the range $\left[v + 2^i, v + 2^{(i+1)}\right), 0 \le i < k$.

In the discussion that follows, we refer to small-world networks specifically as overlays with Property 1.

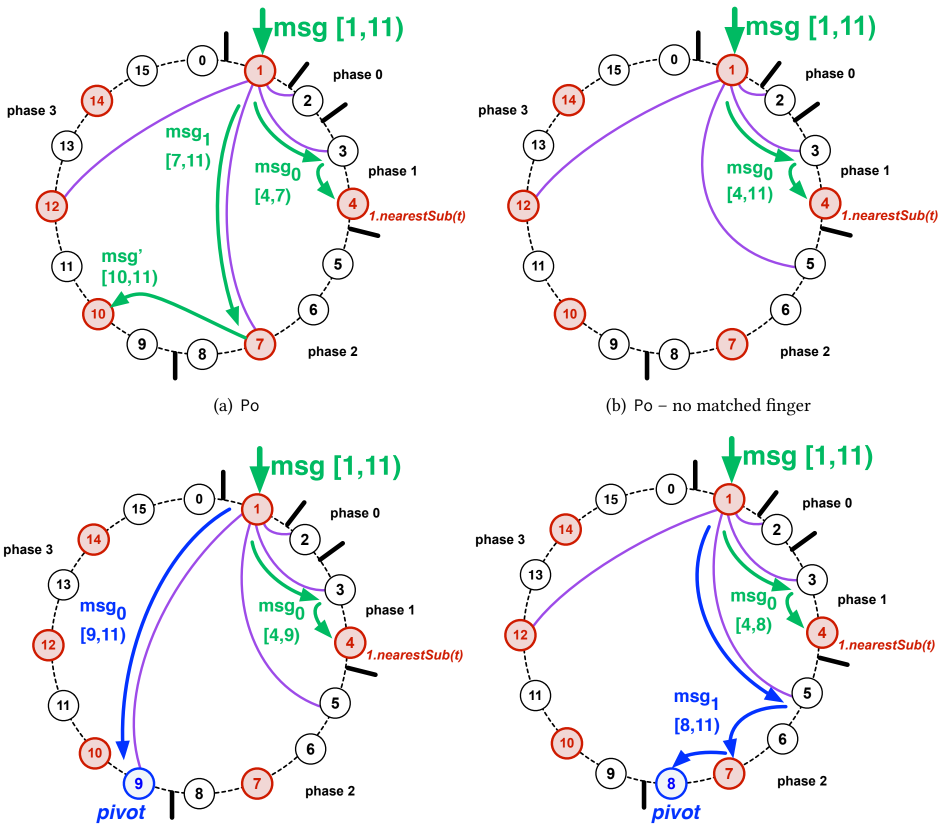
3.2 Greedy algorithms to build overlay

[1] We can also define $\texttt{swDistance}(v, w)$ as the absolute distance between v and w, i.e., minimum of the clockwise and counter-clockwise distances. This difference only affects the constant factors hidden behind the Big-O notations [17, 25].



(a) Vertices and interests (b) A circle with TcoSuppR = 1 (c) Small-world ids assigned sequentially (d) Random circle and small-world ids

Figure 1: Example of GSwicoS



(a) Po (b) Po – no matched finger

(c) Pal or Pie with *pivotByFinger*() (d) Pal with *pivotInDistantRange*()

Figure 2: Examples of BeaConvey with different getNextHops() implementations

We focus on overlay design that best suits pub/sub routing protocols. First, the overlay should be a small-world network. Second, we want the overlay to yield minimum *routing overhead*, which this work and many others [5, 12, 15, 16, 28, 29, 35] define to be the total number of pure forwarding messages, i.e., the messages that each node $v \in V$ receives yet has no interest in.

Intuitively, diminishing the routing overhead prefers *interest closeness*, which tends to place nodes with similar interests topologically close to each other. Topic-connected overlay (TCO) is one way to realize interest closeness, which promises to be capable of eliminating the routing overhead in pub/sub [11, 12]. Informally speaking, TCO organizes all nodes interested in the same topic in a directly connected dissemination sub-overlay. TCO can support the transmission of publications on each topic to all subscribers without using non-interested nodes as intermediate relays. Pub/sub routing atop TCOs saves bandwidth and computational resources otherwise wasted on forwarding and filtering out unwanted messages. Unfortunately, TCO and Property 1 are at odds with each other. More specifically, constructing a TCO with a fixed average or maximum node degree is an NP-hard problem [11, 27], while Property 1 strictly restricts each node to possess a bounded number of small-world fingers. We decide to stick to Property 1 but compromise the TCO requirement; more specifically, we approximate a TCO (i.e., building an overlay as close to a TCO as possible) under the constraint of being a small-world network.

Formally speaking, given an instance (V, T, I), we regard the pub/sub overlay as an undirected graph $G = (V, E)$ over the node set V with the edge set $E \subseteq K$, where we denote by K the ground

Algorithm 1: Greedy heuristic for SWICO (GSwicoSL)

```
GSwicoSL($V$, $T$, $I$)
Input: ($V$, $T$, $I$)
Output: $E$, which forms a small-world and interest-close overlay
 1: $C \leftarrow$ GSwicoS($V$, $T$, $I$) // build short links to order all nodes
 2: $E \leftarrow$ GSwicoL($V$, $T$, $I$, $C$) // build long links for small world
 3: return $E$

▷ GSwicoS($V$, $T$, $I$)
Input: ($V$, $T$, $I$)
Output: $C$, a circle that goes through all nodes
 1: $C \leftarrow \emptyset$
 2: while $C$ does not form a circle for $V$ do
 3:     $P \leftarrow \{e | e$ contributes to a circle for $V$ wrt $C\}$ // Potential edge set
 4:     $e' \leftarrow \arg\max_{e \in P} contrib(e)$
 5:     $C \leftarrow C + e'$
 6: assign small-world ids based on a sequential order of $C$
 7: return $C$

▷ GSwicoL($V$, $T$, $I$, $C$)
Input: ($V$, $T$, $I$, $C$)
Output: $E$, which forms a small-world network for $V$
 1: $E \leftarrow C, P \leftarrow V \times V$
 2: while $P \neq \emptyset$ do
 3:     $e' \leftarrow \arg\max_{e \in P} contrib(e)$
 4:     $E \leftarrow E + e'$
 5:     $P \leftarrow P - e'$
 6:     for all $e : v$.swPhase($e$) = $v$.swPhase($e'$) where $v \in e$ do
 7:         $P \leftarrow P - e$
 8: return $E$
```

set of all possible edges among V, i.e., $K = V \times V$. Given $G = (V, E)$, the sub-overlay *induced* by $t \in T$ is a subgraph $G^{(t)} = (V^{(t)}, E^{(t)})$ such that $V^{(t)} = \{v \in V | I(v, t) = 1\}$ and $E^{(t)} = \{(v, w) \in E | v \in V^{(t)} \wedge w \in V^{(t)}\}$. A topic-connected component (TCC) on topic $t \in T$, is a maximal connected subgraph in $G^{(t)}$. If $G^{(t)}$ contains only one TCC for each topic $t \in T$, then $G = (V, E)$ forms a *topic-connected overlay* (TCO) for (V, T, I).

Given an edge set $E \subseteq K$, $TCC(E)$ stands for the total number of TCCs in $G = (V, E)$ over all topics T,

$$TCC(E) = \sum_{t \in T} \left(\#\text{TCCs in } G^{(t)} = (V^{(t)}, E^{(t)})\right) \tag{1}$$

By definition,

$$TCC(\emptyset) = \sum_{t \in T} \left|V^{(t)}\right| \tag{2}$$

$$TCC(K) = \left|\{t \in T : V^{(t)} \neq \emptyset\}\right| \tag{3}$$

$$TCC(E) = TCC(K) \text{ iff } E \subseteq K \text{ forms a TCO} \tag{4}$$

We can use $TCC(E)$ to measure the progress towards TCO: suppose E is initially empty and grows by adding edges one by one, then $TCC(E)$ starts from $TCC(\emptyset)$ and strictly decreases with every edge addition down to an absolute limit, i.e., $TCC(K)$.

Further, we define the *contribution* (towards TCO) of edge e with respect to a given edge set E to be the number of TCCs that would be reduced by adding e upon E.

$$contrib_E(e) = TCC(E) - TCC(E + e) \tag{5}$$

Given an instance (V, I, T) and an edge set $E \subseteq K$, we place an arbitrary order on all edges in E, i.e., $E = \{e_1, e_2, \ldots, e_m\}$. Let $E_0 = \emptyset$ and $E_i = \{e_1, ..., e_i\}, 1 \leq i \leq m$, then E forms a TCO iff

$$\sum_{1 \leq i \leq m} contrib_{E_{i-1}}(e_i) = TCC(\emptyset) - TCC(K) \tag{6}$$

As Eq.(6) shows, $TCC(\emptyset) - TCC(K)$ represents the amount of TCCs that $E \subseteq K$ should reduce to achieve a TCO. We define the *TCO support ratio* [9] for an overlay edge set E:

$$TcoSuppR(E) = \frac{TCCs(\emptyset) - TCCs(E)}{TCCs(\emptyset) - TCCs(K)}, \forall E \subseteq K \tag{7}$$

TcoSuppR can indicate overlay quality in terms of interest closeness: the higher $TcoSuppR(E)$ is, the closer E approximates a TCO. More specifically, (1) $TcoSuppR(E) \in [0, 1]$; (2) $TcoSuppR(\emptyset) = 0$, and $TcoSuppR(E) = 1$ iff E forms a TCO; and (3) $TcoSuppR(E)$ is monotonically increasing as E expands, i.e.,

$$TcoSuppR(E) \leq TcoSuppR(F), \text{ if } E \subseteq F$$

TcoSuppR turns out to be an effective optimization objective to guide pub/sub overlay design that aims at interest closeness. Two greedy algorithms, GPA and GPM, achieve constant approximation ratios for the NP-hard problems of optimizing *TcoSuppR* under the average or maximum node degree constraint, respectively [9]. Hence, we use *TcoSuppR* to formalize the problem of designing *small-world and interest-close overlay* (SWICO) as follows:

Problem 1. *Given an input instance (V, T, I), construct an edge set $E \subseteq K$ that maximizes TcoSuppR while forming a small-world network with Property 1.*

To tackle Problem 1, we construct a SWICO in two steps: Step I orders all nodes in a ring and assigns a small-world ID to each node according to its relative positions on the ring; Step II selects additional small-world fingers to meet Property 1. In other words, Step I and II are responsible for *short-* and *long*-range small-world fingers, respectively. Recall that the short-range finger of node v is $v.swFinger[0]$, the clockwise immediate neighbour in a small-world network. The long-range fingers are in the i-th small-world phase, $i \geq 1$. It is challenging to maximize the TCO support in both steps, and we adopt a greedy algorithmic design, which is proven to be effective in many TCO design problems [9, 11, 27].

To operationalize the above thoughts, we devise GSwicoSL in Alg. 1, a greedy heuristic for Problem 1. GSwicoSL has two subroutines: GSwicoS for Step I and GSwicoL for Step II, i.e., Line 1 and Line 2 of Alg. 1, respectively.

GSwicoS initializes $C = \emptyset$ and adds to C edge by edge until C forms a circle among all nodes. Each time, Lines 3 of GSwicoS first computes a potential edge set P, which merely contains edges that would extend the current edge set C to a full circle. Formally speaking, $e \in P$ iff there exists a full circle that covers $e + C$. Then Line 4 greedily selects an edge with the greatest contribution towards TCO from P. Once a circle is completed, Line 6 assigns an ID for each node based on its relative position on the C. Fig. 1 illustrates how GSwicoS works with an example. Fig. 1(a) shows an input instance with nodes and their interests. In Fig. 1(b), GSwicoS builds a full circle with the linear ordering of $A{-}B{-}C{-}D{-}E{-}F{-}A$, and this circle attains a *TcoSuppR* of 100%. In Fig. 1(c), GSwicoS assign IDs along the circle. Fig. 1(d) shows another ID assignment along a different circle $A - C - F - D - B - E - A$, which amounts to zero contribution towards TCO, i.e., $TcoSuppR = 0$.

GSwicoL builds *long*-range fingers on top of the circle that Step I produces. GSwicoL starts from $E = C$ and constructs edges iteratively until E attains a small-world network for the given input instance. At each iteration, GSwicoL greedily selects an edge with the highest contribution towards TCO from a potential edge set P,

Data Structure 2: Publication message structure

```
MESSAGE: a publication message transmitting between nodes
  // publication information
  ○ topic: topic associated with the publication
  ○ content: content of the publication
  // internal fields for pub/sub routing
  ○ low: the lowest point of the targeted range, inclusive
  ○ high: the highest point of the targeted range, exclusive
```

Data Structure 3: Routing table at node $v \in V$

```
▷ $v.swFinger$: the finger list indexed by the small-world phase in the
  id space, $v.swFinger[i] \in \left[v + 2^i, v + 2^{(i+1)}\right)$, $0 \le i < \log |V|$.
▷ $v.nearestSub$: a hashtable that maps each topic $t \in T$ to
  $v.nearestSub(t)$, the subscriber of $t$ that is nearest to $v$ clockwise on
  the circle.
```

which enforces the small-world network constraint of Property 1 in Lines 5-7. In particular, after adding e', GSwicoL removes from P all edges that belong to the same small-world phase as e', because there is only one spot for each small-world phase. In Fig. 2(a), we have a small-world network of 16 nodes, i.e., $V = \{0, 1, \ldots, 15\}$. The figure highlights the small-world fingers of node 1, which connects to node 2, 3, 7, and 9, respectively at each phase. In this example, suppose GSwicoL adds edge $(1, 7)$ at some iteration, then we need to remove from P some edges, such as $\{(1, 5), (1, 6), (1, 8)\}$, which are also in phase 2 of node 1.

GSwicoSL (i.e., GSwicoS and GSwicoL) follows the same greedy strategy as GPA and GPM, which we can generalize as follows: the algorithm starts from an initial edge set (usually $\emptyset$) and constructs E iteratively, until reaching some specific termination conditions; at each iteration, the algorithm greedily adds to E from a carefully computed potential edge set P an edge e' with the highest contribution (see Lines 3-4 of GSwicoS and GSwicoL, respectively). We favor this greedy algorithmic design for Problem 1, because its effectiveness has been proven – GPA and GPM are the best-known algorithms to maximize TCO support for their targeted problems [9].

Alg. 1 inherits many nice properties from GPA and GPM, because they share an identical algorithmic framework.

LEMMA 3.1. *Alg. 1 outputs an edge set E for Problem 1 with running time* $O(|V|^2|T|)$.

Many outstanding issues are still unknown and under exploration, such as the complexity of Problem 1, the approximation ratio of Alg. 1, and so on. Fortunately, a greedy heuristic as simple as GSwicoSL performs well enough for the deployment of BeaConvey in practice (see evaluation in §5). It is of great potential and research interest to find more efficient algorithms for the overlay topologies of BeaConvey or other pub/sub systems. Our problem formulation and algorithm design can serve as a solid reference and comparison baseline for future research that aims at a comprehensive knowledge of pub/sub overlay design.

4 BeaConvey routing for pub/sub

4.1 Routing framework

Small-world networks provide a coordinate system that can effectively position each node in the ID space (e.g., a circle). This enables us to design pub/sub routing in a *divide-and-conquer* manner: we divide the dissemination range of a message *msg* into several sub-ranges and conquer each sub-range independently by delivering *msg* to all matched nodes in this sub-range.

Protocol 4: BeaConvey routing for pub/sub at node $v \in V$

```
$v$.BeaConvey()
 1: upon receiving a message $msg$
 2:   if $v = msg.low$ then
 3:     $X \leftarrow$ getNextHops($msg$) // see Protocol 5
 4:     if $|X| > 0$ then
 5:       for all $j = 0, \ldots, |X| - 1$ do
 6:         $msg_j \leftarrow$ clone $msg$
 7:         $msg_j.low \leftarrow X_j$
 8:         if $j + 1 < |X|$ then
 9:           $msg_j.high \leftarrow X_{j+1}$
10:         else
11:           $msg_j.high \leftarrow msg.high$
12:         $send(msg_j)$
13:   else
14:     $send(msg)$
$v.send(msg)$ // forward $msg$ to $v$'s finger nearest to $msg.low$
 1: forward $msg$ to $w$, the finger of $v$ that is nearest to $msg.low$, i.e.,
    $w = \arg\min_{u \in v.swFinger}$ swDistance($u$, $msg.low$)
```

Protocol 5: Get next-hops for a message at node $v \in V$

```
I. $v$.Po($msg$) // NSMF with Pivot None
 1: $t \leftarrow msg.topic$
 2: $X \leftarrow$
    $\left\{v.nearestSub(t) \cup \{u \in v.swFinger | I(u, t)\}\right\} \bigcap [msg.low, msg.high)$
    // initialize $X$, the next-hop array
 3: sort $X$ in ascending order wrt. small-world distances from $v$
 4: return $X$
II. $v$.Pal($msg$) // NSMF and Pivot always
 1: $X \leftarrow$ Po($msg$)
 2: if $(d = |X| > 0) \wedge (X_{d-1} \notin distantRange(msg))$ then
 3:   $pivot \leftarrow pivotByFinger(msg)$
 4:   if $pivot$ = NIL then
 5:     $pivot \leftarrow pivotInDistantRange(msg)$
 6:   append $pivot$ to $X$
 7: return $X$
III. $v$.Pie($msg$) // NSMF and Pivot if exist
 1: $X \leftarrow$ Po($msg$)
 2: if $(d = |X| > 0) \wedge (X_{d-1} \notin distantRange(msg))$ then
 3:   $pivot \leftarrow pivotByFinger(msg)$
 4:   if $pivot \neq$ NIL then
 5:     append $pivot$ to $X$
 6: return $X$
/* Other helper functions */
$v.pivotByFinger(msg)$ // get pivot by a distant-range finger
  return $p \in (distantRange(msg) \cap v.swFinger)$
$v.pivotInDistantRange(msg)$ // get pivot in the distant range
  return $q \in distantRange(msg)$
$v.distantRange(msg)$ // the distant range of $msg$
  return $\left[\frac{msg.low+msg.high}{2}, msg.high\right)$
```

Before proposing our pub/sub routing protocols on small-world networks, we first introduce Data Structure 2 and 3.

Data Structure 2 defines the publication message for our pub/sub routing. Given a message *msg*, fields *msg.topic* and *msg.content* describe publication data. Fields *msg.low* and *msg.high* jointly define the targeted range of *msg*, i.e., *msg* should reach all matched subscribers in the range of $[msg.low, msg.high)$. For example, an initial publisher at $v \in V$ always specifies the target of the publication message to span the whole identifier space, i.e., $[v, v)$.

Data Structure 3 presents the local routing table at each node. First, node v keeps a list of small-world fingers, $v.swFinger$, which corresponds to Property 1; §3 discusses how to construct these fingers. Second, node v maintains in $v.nearestSub$ the nearest subscriber for each topic $t \in T$; we assume that a membership service is available to provide such knowledge. Please note such membership service only requires each node to maintain $\Theta(|T|)$ states, whereas a full interest-aware membership costs a significantly higher space complexity at each node: $\sum_{v \in V} |\{t | I(v,t)\}| = O(|V||T|)$. The implementation details are beyond the scope of this work.

Protocol 4 specifies the BeaConvey pub/sub routing framework atop small-world networks with Property 1. Upon receiving a message *msg* at node v, Protocol 4 delivers *msg* to all matched subscribers in the targeted range $[msg.low, msg.high)$. Line 2 checks whether v is $msg.low$, the first destination of *msg*: if not, Line 14 simply invokes $send(msg)$, which forwards *msg* to v's finger that is nearest to $msg.low$; if $v = msg.low$, then Lines 3-12 spread this message to all subscribers in the targeted range. In Line 3, function `getNextHops`(msg) outputs X, the next-hop array (for *msg*) that is sorted in ascending order according to small-world distances from v; hence, the first next-hop X_0 is always $v.nearestSub(t)$ as long as X is nonempty, where $t = msg.topic$. There are numerous ways to implement `getNextHops()`, and we leave it as a *virtual* function, which should be overwritten by a concrete next-hop selection (see §4.2). Lines 4-12 prepare a message msg_j for every next-hop X_j, $j = 0, \ldots, |X| - 1$. Each X_j is in charge of a sub-range as specified in msg_j, and the union of all these sub-ranges spans the entire targeted range of *msg* $[msg.low, msg.high)$: X_0, ..., X_{d-2}, and X_{d-1} cover $[X_0, X_1)$, ..., $[X_{d-2}, X_{d-1})$, and $[X_{d-1}, msg.high)$, respectively, where $d = |X|$. Note that we skip $[msg.low, X_0) = [v, v.nearestSub(t))$, because this sub-range contains no subscriber of t. Following Line 12, X_j will receive msg_j from v and recursively invoke Protocol 4. This recursive process continues, until the publication message reaches all subscribers in the targeted range.

In summary, Protocol 4 accomplishes pub/sub routing by disseminating the publication message to all subscribers in the targeted range.

4.2 Getting next hops - pivot or not

We instantiate the virtual function `getNextHops()` in Line 3 of Protocol 4. Two optimization objectives guide our design:

(a) *routing overhead*: the total number of messages purely forwarded, i.e., messages that some nodes receive yet has no interest in. We also use this metric for SWICO design in §3.2.

(b) *propagation delay*, which is captured by the average or maximum path length across all message deliveries. The path length of a message *msg* is defined as the number of overlay hops for *msg* to travel from source to destination $msg.low$. BeaConvey operates on top of the application layer, and messages go through a stack of layers for each hop, especially when we rely on secure channels.

Protocol 5 presents three implementations of getting next-hops, i.e., `Po`, `Pal`, and `Pie`. `Po` solely optimizes routing overhead. `Pal` aims at minimizing propagation delay, which unfortunately increases routing overhead. We further design `Pie`, a reasonable middle ground between the previous two, that treads the balance between both optimization objectives.

4.2.1 `Po`

Function `Po`(msg) specifies the algorithm of Nearest Subscribers and Matched Fingers (NSMF) with Pivot None . Line 1 sets $t = msg.topic$, and Line 2 computes the next-hop array X by combining (1) the nearest subscriber for t and (2) all small-world fingers that subscribe to t, where all next-hops lie in the targeted range of *msg*. Line 3 sorts all next-hops in X in ascending order according to the small-world distances from v, and thus X_0 is always $v.nearestSub(t)$ as long as X is nonempty.

In Fig. 2(a), we have a small-world network of 16 nodes, i.e., $V = \{0, 1, \ldots, 15\}$. The figure highlights the small-world fingers of node 1, which connects to node 2, 3, 7, and 9, respectively at each phase. For some topic $t \in T$, nodes that subscribe to t are $V^{(t)} = \{1, 4, 7, 10, 12, 14\}$, i.e., shaded vertices in Fig. 2(a). Suppose that node 1 receives a message *msg* on t with the targeted range $[1, 11)$, which we simply denote as $msg\,[1, 11)$. Node 1 should send *msg* to all nodes that are interested in t and in the range of $[1, 11)$, i.e., $\{4, 7, 10\}$. After executing `Po`(msg), node 1 obtains the sorted next-hop array $X = \{4, 7\}$: $X_0 = 4$ is the nearest subscriber for t, and $X_1 = 7$ is a matched finger; X excludes other fingers in $[1, 11)$ (such as node 2, 3, 9), because they are not interested in t. Node 1 prepares $msg_0\,[4, 7)$ for $X_0 = 4$ and $msg_1\,[7, 11)$ for $X_1 = 7$, which hierarchically partitions the targeted range of $[1, 11)$ into multiple successive sub-ranges. Since node 1 has no finger linking to $X_0 = 4$, it sends msg_0 to node 3, which then forwards msg_0 to node 4. Node 1 directly sends msg_1 to node $X_1 = 7$. Upon receiving msg_0 and msg_1, both node 4 and 7 call Protocol 4, recursively. Node 4 sends no further messages, since its nearest subscriber (node 7) is out of the targeted range of $msg_0\,[4, 7)$. Node 7 sends a single message $msg'\,[10, 11)$ to its nearest subscriber 10. The recursive processes end at node 10, since node 10 is the only one in the targeted range of $msg'\,[10, 11)$.

BeaConvey with `Po` (a.k.a. BeaConvey(`Po`)) relies little on pure forwarders, which can only occur when some node sends a message to the nearest subscriber, e.g., node 3 in Fig. 2. However, the propagation delay of a message (i.e., the number of hops from source to destination) may be linear in the number of nodes. Fig. 2(b) is almost the same as Fig. 2(a) except that node 1 chooses 5 instead of 7 in its 2nd small-world phase. In Fig. 2(b), `Po`(msg) returns $X = \{4\}$, because node 1 has no matched finger for t. It is possible that, at each recursive invocation of BeaConvey, `Po` obtains nothing but the nearest subscriber in the next-hop array; in this case, it takes $O(|V|)$ relays for *msg* to reach the furthermost subscriber.

4.2.2 `Pal`

To improve the propagation delay, we devise NSMF and Pivot Always (`Pal`) in Protocol 5. `Pal` extends `Po` by always securing in the next-hop array a *pivot* – a node in the *distant range* of *msg*, which $distantRange(msg)$ defines as the second half of the targeted range in the farther end, i.e., $\left[\frac{msg.low+msg.high}{2}, msg.high\right)$. With a pivot, `Pal` guarantees to at least halve the targeted range at each recursive call of BeaConvey. Therefore, BeaConvey (`Pal`) yields a maximum path length of $O(\log |V|)$. More specifically, `Pal` first checks whether there exists a pivot in the next-hop array output by `Po` in Line 2. Since X is already sorted, we just need to examine whether the last node of X is in the distant range of *msg*. `Pal` finds one pivot if X contains none. Line 3 calls $pivotByFinger()$, trying to assign the pivot by an existing finger in the distant range of *msg*. If no such finger exists, then Line 5 pivots at some node in the distant range of *msg* by function $pivotInDistantRange()$.

In Fig. 2(c), `Pal` appoints finger 9 to be the pivot, since node 9 resides in $[6, 11)$, the distant range of $msg[1, 11)$. While Fig. 2(c) and Fig. 2(b) share the same overlays, `Pal` appends a pivot beyond the output of `Po`.

In Fig. 2(d), although node 1 contains no distant-range finger for $msg[1, 11)$, `Pal` still enforces a pivot at node 8.

Table 1: Pub/Sub systems that we evaluate

BeaConvey(O, R)	BeaConvey with overlay O and routing R
Scribe(O)	Rendezvous routing on the small-world overlay O
iScribe(O)	*Inverted* Scribe on the small-world overlay O

Table 2: Types of small-world overlays

SL	Short fingers by GSwicoS and Long fingers by GSwicoL
S	Short fingers by GSwicoS and Long fingers by Chord or RandomL
L	Short fingers by RandomS and Long fingers by GSwicoL
–	Short fingers by RandomS and Long fingers by Chord or RandomL

Pal improves the worst-case propagation delay from $O(|V|)$ to $O(\log |V|)$. However, the additional pivots incur more routing overhead than Po for forwarding irrelevant messages. In Fig. 2(d), it requires two pure forwarders (i.e., $\{5, 8\}$) to deliver $msg_1[8, 11)$ from node 1 to pivot 8.

4.2.3 **Pie**

To overcome the shortcomings of Po and Pal, we design an algorithm in the middle ground, NSMF and Pivot If Exist, Pie for short. As shown in the third function of Protocol 5, Pie adds a pivot only if there exists a finger in the distant range of the message. Similar to Pal, Pie performs the pivot check for the next-hop array and then attempts to find a pivot if there is none. While Pal searches for a pivot within the whole distant range of *msg*, Pie merely focuses on $v.swFinger \cap distantRange(msg)$: if node v has a distant-range finger for the incoming message, then Pie follows the steps of Pal; otherwise, Pie simply proceeds without a pivot, just like Po.

In Fig. 2(c), Pie and Pal exhibit the same behavior: assigning the pivot by node 9, an existing finger.

In Fig. 2(d), Pie behaves like Po and does not take a pivot, since $v.swFinger \cap distantRange(msg) = \emptyset$. In contrast, Pal aggressively chooses a pivot anyway at the expense of more pure forwarders.

5 Evaluation

5.1 Experiment setup

We evaluate BeaConvey and other pub/sub systems (see Table 1) using PeerSim [26].

We choose Scribe [7] for comparison, which represents *rendezvous routing* for pub/sub on small-world networks. Scribe builds a multicast tree for each topic $t \in T$, which associates a rendezvous point to t, denoted by $RP(t)$. The multicast tree of t is rooted at $RP(t)$ and spans all subscribers. This multicast tree may contain pure forwarders, which are not interested in t. Scribe first routes each publication message of t from its source to $RP(t)$ and then disseminates the message from $RP(t)$ along the multicast tree[2].

Further, we develop iScribe, inverted Scribe. The original Scribe builds each multicast tree by routing from every subscriber to the rendezvous point. This makes sense if the subscribers are scattered all over the network. However, long-range links from each subscriber to rendezvous point do not have much chance to share, even if these subscribers stay close to each other. Thus, we build the multicast trees of iScribe in an opposite direction: from rendezvous point to each subscriber. In this way, if subscribers stay closer, the probability of sharing links is higher. Systems like Bayeux [41] adopt iScribe as their underlying routing protocols.

Our proposed BeaConvey is equipped with two parameters: overlay O (see Table 2) and routing R (i.e., Po, Pal, or Pie in Alg. 5). Meanwhile, Scribe and iScribe just need to specify overlay O.

Table 2 lists four types of overlays based on how we build short and long fingers for small-world networks. We develop pub/sub overlays by GSwicoSL in Alg. 1 and use the following naive overlays as comparison baselines:

- short fingers: RandomS, an arbitrary circle that randomly assigns small-world ids and therefore randomly determines the short-range fingers.
- long fingers: Chord [36] or RandomL:

(1) Chord: a DHT that always selects the finger as the first node in each small-world phase, i.e., $v.swFinger[i] = (v + 2^i)$, $0 \le i < \log |V|$, $\forall v \in V$.
(2) RandomL, a small-world network that chooses the long-range finger randomly in each of the small-world phases: all nodes in the range $\left[v + 2^i, v + 2^{i+1}\right)$ are equally probable to become $v.swFinger[i]$, $0 \le i < \log |V|$, $\forall v \in V$.
All types of overlays in Table 2 preserve Property 1. These overlays have the same node degrees, and the expected path length to route between any two nodes is $O(\log |V|)$ [17]. However, GSwicoS strives to optimize interest-closeness for short fingers, whereas RandomS do not. GSwicoL attempts to optimize interest-closeness for long fingers, whereas Chord and RandomL do not.

We evaluate System (e.g., BeaConvey, Scribe, or iScribe) with regards to the two basic optimization metrics (see definitions in §4.2): (a) the routing overhead $\mathbb{R}_{\text{System}}$ (the sum of irrelevant messages that each node has to forward) and (b) propagation delay, which is measured by $\mathbb{AP}_{\text{System}}$ and $\mathbb{MP}_{\text{System}}$, the average and maximum path lengths across all messages. We also look at *TcoSuppR* in Eq. (7) for different pub/sub overlays.

5.2 Experiment workloads

We synthetically generate pub/sub subscriptions with three types of topic popularities: uniform, Zipf, and exponential. We also extract data from Facebook and Twitter.

(1) *Synthetic workloads*: We generate input instances (V, T, I) as follows: $|V| \in [1\,000, 10\,000]$, $|T| \in [1\,000, 10\,000]$, and each node $v \in V$ has a fixed subscription size $|\{t \in T | I(v, t)\}| = 20$. We associate each topic $t \in T$ with $p(t)$, and each node subscribes to t with probability $p(t)$. The value of $p(t)$ follows either a uniform, a Zipf (with $\alpha = 2.0$), or an exponential distribution, which we call Unif, Zipf, or Expo, respectively. These distributions approximate actual workloads used in industrial pub/sub today [11, 12, 24, 38].

(2) *Facebook dataset*: We use a Facebook dataset [39] with over 3 million distinct user profiles and 28.3 million social relations. In Facebook, when a user, say Alice, performs an activity (e.g., updating her status, sharing photos, or commenting on a post), all Alice's friends receive a notification. Therefore, we model each user as a topic, and all her friends as the respective subscribers. Likewise, the friend set of Alice forms her subscription set. Facebook relations are bidirectional: friends in Alice's social graph subscribe to notifications about Alice and vice versa. (Note that this is a simplification as nowadays a Facebook user can unfollow her friend or define an asymmetric policy.)

(3) *Twitter dataset*: We take a public Twitter dataset [22], containing 41.7 million distinct user profiles and 1.47 billion social followee/follower relations. Like Facebook, we model users as topics

[2] The original Scribe caches the addresses of the rendezvous points for subsequent multicasts for the same topics. Similarly, it is straightforward to apply this technique to locate the nearest subscribers in BeaConvey. However, we switch off these features to enforce fairness while better revealing the overlay qualities and routing efficiency. In our evaluation, BeaConvey, Scribe and all others only transmit messages on existing overlay links.

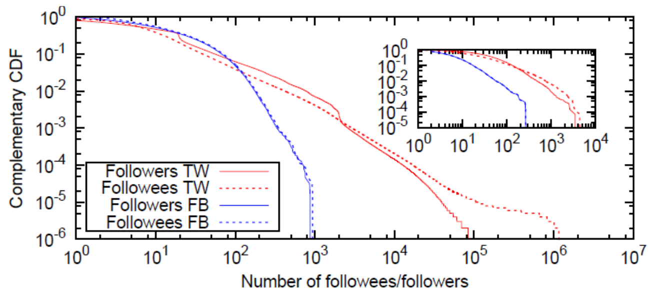


Figure 3: Complementary cumulative distribution function (CCDF) of followee/follower counts. *Outer*: Facebook (3M users) and Twitter (41.7M users). *Inner*: FB 10K and TW 10K.

Table 3: Performance ratio against Scribe(-) under Facebook

System	routing overhead		avg. path length	
	FB1K	FB10K	FB1K	FB10K
BeaConvey(SL, Pie)	0.266	0.449	0.700	0.968
BeaConvey(S, Pie)	0.355	0.612	0.902	1.629
BeaConvey(L, Pie)	0.683	0.827	0.780	1.15
BeaConvey(-, Pie)	0.908	1.127	1.07	2.088
iScribe(SL)	0.427	0.636	0.992	1.035
iScribe(S)	0.428	0.591	0.975	1.007
iScribe(L)	0.993	1.109	0.987	1.024
iScribe(-)	1.007	1.049	1.004	0.9999
Scribe(SL)	0.914	0.908	1.014	1.040
Scribe(S)	1.05	1.024	0.976	1.007
Scribe(L)	0.675	0.819	0.985	1.031
Scribe(-)	1	1	1	1

and subscribers. However, relations in Twitter are unidirectional, e.g., Alice following Bob does not imply that Bob follows back.

We extract the workloads from the original Facebook and Twitter social graphs with a sampling methodology [30, 35]. We start with a few users as seeds and traverse the social graph via breadth first search until reaching the targeted number of nodes, and our sample takes all edges among the visited nodes. We report experimental results for samples of 1K or 10K, i.e., $|V| \approx 1K$, $|T| \approx 1K$, or $|V| \approx 10K$, $|T| \approx 10K$. We denote our sample instances by FB 1K, FB 10K, TW 1K, and TW 10K, respectively. Fig. 3 illustrates that our extracted samples retain properties of the original data sets.

Our publication workloads are uniformly random with respect to the topics and all subscribers. At each iteration, the system publishes one message on some topic $t \in T$ from an issuer node $v \in V$: every topic $t \in T$ possesses the same probability $1/|T|$, and all subscribers of t have the equal chance to become the issuer.

5.3 Experiments for online social networks

We compare the BeaConvey, Scribe, and iScribe families under the FB and TW data sets. Table 3, 4, and 5 report the comparable ratios of each system normalized by Scribe(-), the original Scribe.

5.3.1 Overlay Alg. 1 runs efficiently in practice. It takes around 14.8 and 19.4 minutes to compute SWICO from scratch under FB 10K and TW 10K, respectively. This runtime cost is insignificant as compared to the intervals between successive churn events in a typical data-center-wise pub/sub system, which are usually in the order of tens of hours, depending on the cluster size [10].

Table 4: Performance ratio against Scribe(-) under Twitter

System	routing overhead		avg. path length	
	TW1K	TW10K	TW1K	TW10K
BeaConvey(SL, Pie)	0.573	0.748	0.902	1.131
BeaConvey(S, Pie)	0.617	0.797	1.273	1.252
BeaConvey(L, Pie)	0.757	1.018	0.931	1.121
BeaConvey(-, Pie)	0.853	1.17	1.357	1.310
iScribe(SL)	0.664	0.768	1.041	1.058
iScribe(S)	0.596	0.671	1.020	1.001
iScribe(L)	0.913	1.167	1.007	1.040
iScribe(-)	0.817	1.016	1.008	1.004
Scribe(SL)	0.853	0.894	1.035	1.051
Scribe(S)	1.096	1.012	1.019	0.999
Scribe(L)	0.714	0.841	0.996	1.040
Scribe(-)	1	1	1	1

Table 5: BeaConvey with different routing against Scribe(-)

System	routing overhead		avg. path length	
	FB1K	FB10K	FB1K	FB10K
BeaConvey(SL, Pie)	0.266	0.449	0.700	0.968
BeaConvey(SL, Po)	0.182	0.319	0.794	1.300
BeaConvey(SL, Pal)	0.341	0.557	0.729	0.981
	TW1K	TW10K	TW1K	TW10K
BeaConvey(SL, Pie)	0.573	0.748	0.902	1.131
BeaConvey(SL, Po)	0.359	0.489	1.145	1.688
BeaConvey(SL, Pal)	0.712	0.917	0.928	1.179

We concentrate on the impact of the overlay topologies upon different systems. We fix Pie as BeaConvey's routing protocol, because Pie achieves the best performance among all three routing schemes (see §5.3.2 and §5.5).

Table 3 and 4 show that BeaConvey(SL,Pie) substantially outperforms Scribe(−) or iScribe(−) in routing overhead. The superiority of BeaConvey(SL,Pie) over Scribe(−) and iScribe(−) stems from both improved overlays and better routing protocols. Under FB 1K, $\mathbb{R}_{\text{BeaConvey(SL,Pie)}} = 0.266 \cdot \mathbb{R}_{\text{Scribe}(-)}$, resulting in a total reduction of $(1 - 0.266) = 0.734 \cdot \mathbb{R}_{\text{Scribe}(-)}$. First, an improved overlay amounts to an routing overhead reduction of $(1 - 0.427) = 0.573 \cdot \mathbb{R}_{\text{Scribe}(-)}$, where 0.427 is ratio of iScribe(SL) against Scribe(−), the lowest one among all different overlays in Scribe and iScribe families. Second, the routing protocol of BeaConvey with Pie further earns $(0.734 - 0.573) = 0.161 \cdot \mathbb{R}_{\text{Scribe}(-)}$ over the rendezvous routing strategies of Scribe or iScribe. In other words, for all routing overhead improvement of BeaConvey(SL,Pie) against Scribe(−), $78\% = 0.573/0.734$ comes from SWICO, and 22% is attributed to more scalable routing.

We take a closer look at BeaConvey. First, SL always produces the lowest routing overhead of BeaConvey among all four types of overlays. This demonstrates that interest-closeness positively impacts the routing overhead. Second, under both FB and TW workloads, optimizing short fingers (i.e., assigning ids) is more important than optimizing long ones, although short fingers are substantially outnumbered. This is clear when we compare BeaConvey(S, Pie) with BeaConvey(L, Pie); besides, BeaConvey achieves significant performance benefits in both routing overhead and message latencies, when the overlay is upgraded from L to SL.

Scribe also benefits from interest-closeness. Unlike BeaConvey, Scribe gains more routing overhead improvement from the optimization of long fingers than short fingers. In Table 4, under TW 1K, Scribe(L) is 71.4% of Scribe(-), whereas Scribe(S) is even higher than Scribe(-) by almost 10%. Recall that Scribe builds the multicast tree for each topic t by routing from every subscriber s to the rendezvous point $RP(t)$, which is not necessarily interested in t. The route path from s to $RP(t)$ consists of two parts:

a. long-range links that originates from s and
b. short-range links that terminates at $RP(t)$.

Long finger optimization leads to less routing overhead in Part a. However, short finger optimization does not always reduce routing overhead in Part b; this is because $RP(t)$ is not a subscriber of t, and nodes that stay close to $RP(t)$ are probably uninterested in t in a circle produced by GSwicoS.

iScribe is opposite of Scribe: interest-closeness in short fingers leads to more routing overhead reduction. iScribe builds the multicast tree by routing from the rendezvous point $RP(t)$ to every subscriber s. In Table 4, under TW 10K, iScribe achieves a reduction of $(1.016 - 0.671) = 0.345 \cdot \mathbb{R}_{\text{Scribe(-)}}$ because of better interest-closeness in short fingers, but optimization in long fingers has a negative impact by an increase of $(1.167 - 1.016) = 15.1\% \mathbb{R}_{\text{Scribe(-)}}$. The route path from $RP(t)$ to s consists of two parts:

c. long-range links that originates from $RP(t)$ and
d. short-range links that terminates at s.

Short finger optimization leads to less routing overhead in Part d. However, long finger optimization does not always reduce routing overhead in Part c; this is because $RP(t)$ is not a subscriber of t, and its long fingers are probably uninterested in t in an overlay produced by GSwicoL.

Reviewing Scribe and iScribe together, we observe that iScribe(S) consistently outperforms Scribe(L). This is aligned with BeaConvey that optimizing short fingers leads to more performance acceleration than optimizing long fingers under FB and TW workloads.

In terms of average path lengths, BeaConvey(SL,Pie) also yields slightly better results as compared to Scribe(-) or iScribe(-) in most cases. In general, the average path lengths (and thus message latencies) are of the same order of magnitude for all systems in Table 3 and 4, as well as the rest of experiment results.

5.3.2 Routing

We look at how different routing strategies impact BeaConvey's performance, where we fix the overlay to be SL.

Table 5 shows that Pie lies between Po and Pal and achieves the best balance between routing overhead and message latency. Po and Pal only optimize on one metric but perform poorly with regard to the other. We therefore only present Pie in most evaluation (as in 5.3.1).

5.4 Impact of overlay

Fig. 4-7 depict how various overlays impact BeaConvey, Scribe, and iScribe as workloads scale up under Zipf and Unif. We let $|V| = 1\,023$, $|T| \in [1\,000, 10\,000]$, and fix Pie for BeaConvey.

5.4.1 Zipf

In general, our experiments using Zipf are in line with the empirical evaluation on Facebook and Twitter in §5.3.

We look at routing overhead. Fig. 4 depicts the routing overhead ratios of different systems against Scribe(-). First, BeaConvey(SL,Pie) substantially outperforms Scribe(-) or iScribe(-). On average, $\mathbb{R}_{\text{BeaConvey(SL,Pie)}} = 0.594\mathbb{R}_{\text{Scribe(-)}}$, i.e., the overhead reduction amounts to $0.406\mathbb{R}_{\text{Scribe(-)}}$. Second, BeaConvey achieves lower routing overhead than Scribe or iScribe even when they have identical overlays, which demonstrates the inferiority of rendezvous routing. Third, BeaConvey gains better routing efficiency by increasing interest closeness in both short and long fingers, and a few short fingers are more influential than many long fingers.

We look at average path lengths in Fig. 5. BeaConvey yields the best average path lengths, even though the gaps between different systems are marginal. For instance, $\mathbb{AP}_{\text{BeaConvey(SL,Pie)}} = 4.88$, $\mathbb{AP}_{\text{Scribe(-)}} = 7.20$, on average.

5.4.2 Unif

Fig. 6 and 7 compare various pub/sub systems under Unif, which is different from Zipf in a number of respects.

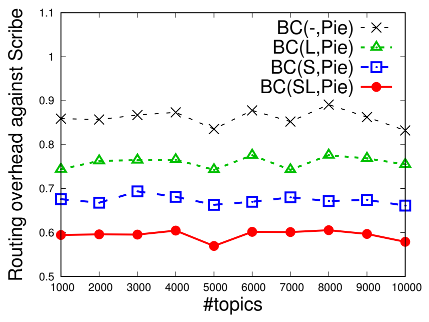
First, Fig. 6 shows that, under Unif, routing turns out to be more contributive than overlay towards the overhead improvement of BeaConvey against Scribe, and this dominance becomes stronger as the workloads scale up. On the one hand, Fig. 6(a) shows that BeaConvey(SL,Pie) outperforms Scribe(-) by wide margins, which becomes even more remarkable as the topic set expands. On the other hand, in Fig. 6(b) and 6(c), Scribe and iScribe family members are of little difference, and they tend toward equality with the raise in the number of topics. At $|T| = 10\,000$, $\mathbb{R}_{\text{BeaConvey(SL,Pie)}} = 0.379\mathbb{R}_{\text{Scribe(-)}}$, an overall reduction of $0.621\mathbb{R}_{\text{Scribe(-)}}$: 94.9% comes from BeaConvey routing with Pie, while 5.1% comes from SWICO. The key rationale behind this is: under Unif, the correlation among nodes is weak, and there is not much to harvest by optimizing interest closeness in the overlay. Under skewed distributions (e.g., FB, TW or Zipf), the overlay can make a considerable difference thanks to the abundance of interest closeness.

Second, BeaConvey benefits slightly more from longer fingers than short ones in terms of both routing overhead and message latencies. In Fig. 6(a), the routing overhead ratios of BeaConvey(L, Pie) and BeaConvey(S, Pie) are about 0.585 and 0.653, respectively on average. The core reason lies again in innate lack of interest closeness in Unif. It is not surprising that each node v has little preference for its neighbor selection on the ring, i.e., locality becomes unimportant. In such scenarios, long fingers carry more weight because they outnumber short fingers. On the contrary, skewed pub/sub workloads bear rich interest closeness so that even a minority of short fingers can make a dominant contribution.

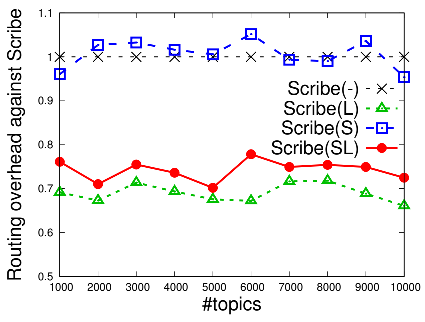
Third, in Fig. 7, BeaConvey exhibits better scalability in path lengths than both Scribe and iScribe, and the relative ratios drop rapidly down to around 0.5 as the number of topics increases. Within each BeaConvey, Scribe, or iScribe family, the gaps between different members shrink as topic set expands. We believe that, the average path lengths for the four BeaConvey instances in Fig. 7(a) will converge to the same value, $\log |V|$, as $|T|$ soars to infinity. SWICO gains benefits over others by exploiting the correlation embedded in the subscriptions of all nodes, and this correlation is diminishing as a result of expanding the topic set. When the number of topics is sufficiently large, any small-world overlay equally possesses zero amount of interest-closeness. In these scenarios, BeaConvey just need to deliver each message from the source to a singleton destination by one round of small-world routing, so the expected path length is about $\log |V|$ hops; meanwhile, Scribe or iScribe requires two rounds of small-world routing, $2 \cdot \log |V|$ hops. Therefore, with an infinite extension of the topic set, the BeaConvey family would converge to 0.5 in Fig. 7(a) , while both Scribe and iScribe would converge to 1 in Fig. 7(b) and 7(c).

5.4.3 Summary

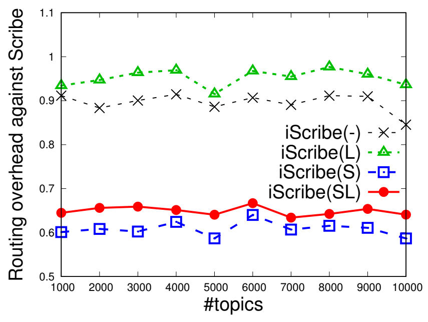
In BeaConvey, overlay and routing complement each other and work together to enhance the system performance under a variety of pub/sub workloads. Under skewed distributions, overlay prevails over routing in making BeaConvey scalable, because richness of interest closeness is available to exploit in the workloads; besides, a well-constructed SWICO exhibits



(a) BeaConvey

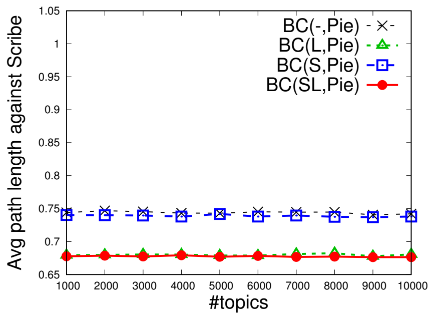


(b) Scribe

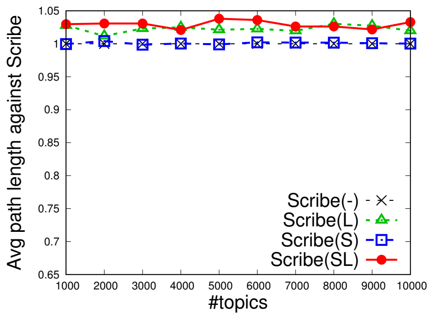


(c) iScribe

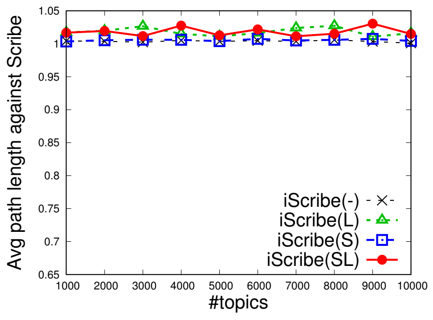
Figure 4: Routing overhead against Scribe: Zipf, $|V| = 1023$, $|T|$ varies



(a) BeaConvey

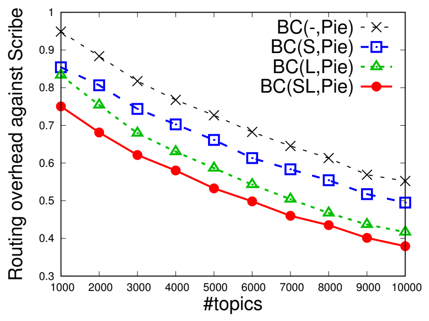


(b) Scribe

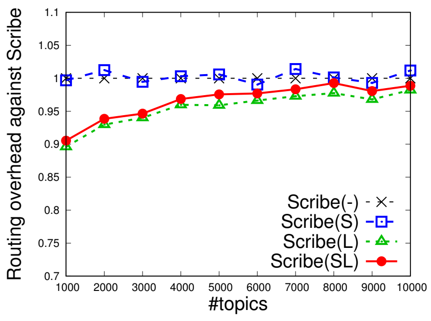


(c) iScribe

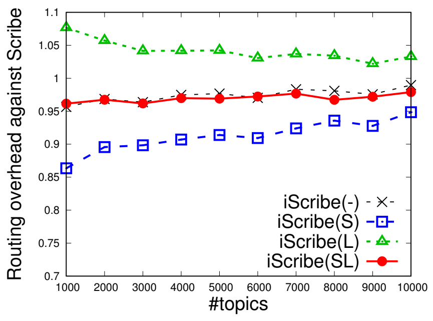
Figure 5: Average path lengths against Scribe: Zipf, $|V| = 1023$, $|T|$ varies



(a) BeaConvey

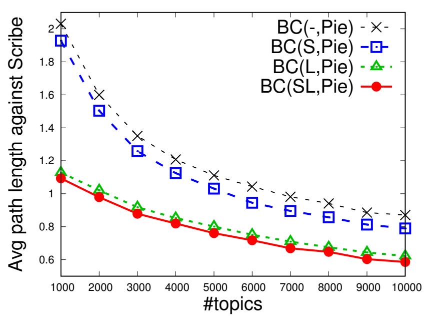


(b) Scribe

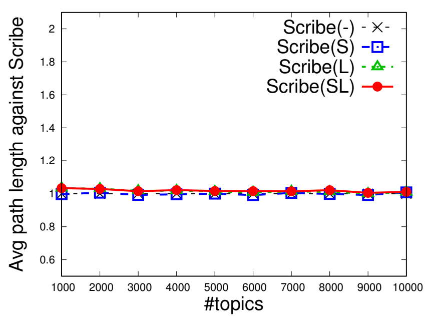


(c) iScribe

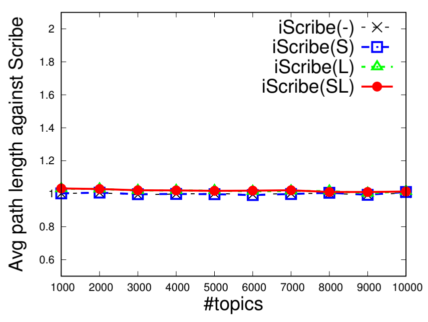
Figure 6: Routing overhead against Scribe: Unif, $|V| = 1023$, $|T|$ varies



(a) BeaConvey



(b) Scribe

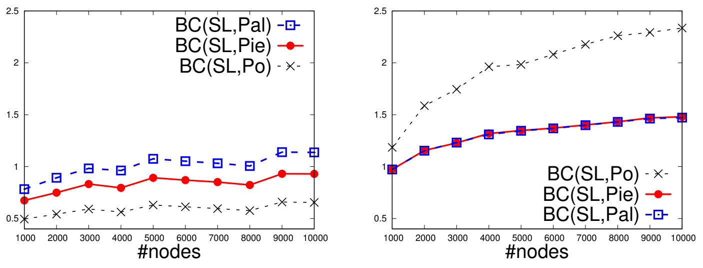


(c) iScribe

Figure 7: Average path lengths against Scribe: Unif, $|V| = 1023$, $|T|$ varies

the *Pareto principle*: over 80% of the benefits is supported by fewer than 20% of all overlay connections, i.e., short fingers. Under non-skewed distributions, routing becomes increasingly dominant for performance enhancement due to the lack of interest closeness; short fingers also lose their vital roles in overlay optimization because size matters.

5.5 Impact of routing



(a) Routing overhead (b) Avg hops

Figure 8: BeaConvey against Scribe: Unif, $|V|$ varies, $|T| = 2000$

Fig. 8 compare BeaConvey with all three getNextHops() implementations under Unif as the number of nodes $|V|$ increases from $1,000$ to $10,000$, where we fix $|T| = 2,000$.

Fig. 8(a) plots the routing overhead ratios between BeaConvey and Scribe(-). All three BeaConvey protocols outperform Scribe(-), as their corresponding curves are under the horizontal line of 1. Among the three instances of BeaConvey, Po achieves the best routing overhead, which is 59.2% that of Scribe(-) on average. Pal renders the worst with a relative ratio of 100.6%, because Pal guarantees a pivot at each branching, which may rely on unmatched fingers to forward irrelevant messages. This disadvantage of Pal motivates us to design Pie, which reduces the routing overhead down to 83.5% of $\mathbb{R}_{\text{Scribe}(-)}$.

Fig. 8(b) depicts the average path lengths for different BeaConveyprotocols as compared to Scribe. Despite the credits in the routing overhead, Po produces the longest path lengths among all: the average path length of BeaConvey(SL, Po) is $1.96 \cdot \mathbb{AP}_{\text{Scribe}}$, and the maximum path length of BeaConvey(SL, Po) is $6.61 \cdot \mathbb{MP}_{\text{Scribe}}$, on average. Moreover, the gaps are increasing with $|V|$. This shows that Po is insufficient to meet all design goals, and we need better algorithms for diminishing the propagation delay. Both Pie and Pal perform better with similar outputs. The average path length is around $1.3 \cdot \mathbb{AP}_{\text{Scribe}}$ for both. The maximum path lengths of Pie and Pal are 3.80 and 3.35 of $\mathbb{MP}_{\text{Scribe}}$. Pie closely approximates Pal in path lengths, which demonstrates the effectiveness of selective pivoting.

6 Conclusions

We develop BeaConvey, a data-center-wide distributed pub/sub system on top of small-world networks. BeaConvey presents a streamlined design that combines both overlay topologies and routing protocols in a pub/sub system, whereas most previous pub/sub systems simply place emphasis on one of these two aspects. Thanks to the broader design space that BeaConvey embraces, we reveal a number of fundamental trade-offs among routing overhead, propagation delay, overlay quality, and so on. Empirical evaluation demonstrates that both improved overlays and finer-tuned routing bring remarkable advantages to BeaConvey.

References

[1] Apache Kafka. http://kafka.apache.org/.
[2] Google Cluster Data. http://code.google.com/p/googleclusterdata/.
[3] Hadoop. http://hadoop.apache.org/.
[4] IBM Watson IoT Platform. https://www.ibm.com/internet-of-things.
[5] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. TERA: topic-based event routing for peer-to-peer architectures. In *DEBS*, 2007.
[6] 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.*, 2007.
[7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 2002.
[8] C. Chen and Y. Tock. Design of routing protocols and overlay topologies for topic-based publish/subscribe on small-world networks. In *Middleware Ind. '15*.
[9] C. Chen, Y. Tock, and H.-A. Jacobsen. Overlay design for topic-based publish/subscribe under node degree constraints. In *ICDCS '16*.
[10] C. Chen, R. Vitenberg, and H.-A. Jacobsen. OMen: Overlay Mending for Topic-based Publish/Subscribe Systems Under Churn. In *DEBS 2016*.
[11] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing scalable overlays for pub-sub with many topics: Problems, algorithms, and evaluation. In *PODC*, 2007.
[12] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS*, 2007.
[13] 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.
[14] S. Girdzijauskas. *Designing peer-to-peer overlays: a small-world perspective.* PhD thesis, EPFL, 2009.
[15] S. Girdzijauskas, G. Chockler, R. Melamed, and Y. Tock. Gravity: An interest-aware publish/subscribe system based on structured overlays. In *DEBS*, 2008.
[16] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed. Magnet: practical subscription clustering for internet-scale publish/subscribe. In *Debs'10*.
[17] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Sigcomm'03*.
[18] J. C. Corbett, etc. Spanner: Google's globally-distributed database. In *OSDI'12*.
[19] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann. Self-organizing broker topologies for publish/subscribe systems. In *SAC*, 2007.
[20] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *STOC*, 2000.
[21] J. M. Kleinberg. Small-world phenomena and the dynamics of information. In *NIPS*, pages 431–438, 2001.
[22] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW '10*.
[23] G. Li, V. Muthusamy, and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. In *Middleware '08*.
[24] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. In *IMC*, 2005.
[25] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *USITS*, 2003.
[26] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *P2P*, 2009.
[27] M. Onus and A. W. Richa. Parameterized maximum and average degree approximation in topic-based publish-subscribe overlay network design. In *ICDCS*, 2010.
[28] 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*, 2009.
[29] F. Rahimian, S. Girdzijauskas, A. H. Payberah, and S. Haridi. Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks. In *IPDPS*, 2011.
[30] F. Rahimian, T. Le Nguyen Huu, and S. Girdzijauskas. Locality-awareness in a peer-to-peer publish/subscribe network. In *DAIS*, 2012.
[31] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Sigcomm'01*.
[32] J. Reumann. GooPS: Pub/Sub at Google. Lecture at CANOE Summer School, Oslo, Norway, August 2009.
[33] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware'01*.
[34] V. Setty, G. Kreitz, R. Vitenberg, M. van Steen, G. Urdaneta, and S. Gimåker. The hidden pub/sub of spotify. In *DEBS '13*.
[35] V. Setty, M. van Steen, R. Vitenberg, and S. Voulgaris. Poldercast: Fast, robus, and scalable architecture for p2p topic-based pub/sub. In *Middleware*, 2012.
[36] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *Sigcomm'01*.
[37] M. A. Tariq, B. Koldehofe, and K. Rothermel. Efficient content-based routing with network topology inference. In *DEBS*, 2013.
[38] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In *IASTED PDCS*, 2005.
[39] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *EuroSys '09*.
[40] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2006.
[41] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, 2001.