Eliminating Tight Coupling using Subscriptions Subgrouping in Structured Overlays

Muhammad Shafique
Alumni Carleton University Ottawa Canada
idresi@protonmail.com

ABSTRACT
Advertisements and subscriptions are tightly coupled to generate publication routing paths in content–based publish/subscribe systems. Tight coupling requires instantaneous updates in routing tables to generate alternative paths which prevents offering scalable and robust dynamic routing in cyclic overlays when link congestion is detected. We propose, OctopiA, first distributed publish/subscribe system for content–based inter–cluster dynamic routing using purpose–built structured cyclic overlays. OctopiA uses a novel concept of subscription subgrouping, which divides subscriptions into disjoint sets called subscription subgroups. The purpose–built structured cyclic overlay is divided into identical clusters where subscriptions in each subgroup are broadcast to an exclusive cluster. Our advertisement and subscription forwarding algorithms use subscription subgrouping to eliminate tight coupling to offer inter–cluster dynamic routing without requiring updates in routing tables. Experiments on a cluster testbed with real world data show that OctopiA reduces the number of saved advertisements in routing tables by 93%, subscription broadcast delay by 33%, static and dynamic publication delivery delays by 25% and 54%, respectively.

1 INTRODUCTION
Content–based Publish/Subscribe (pub/sub) systems provide asynchronous dissemination of data in event–based distributed applications [13]. A publisher publishes the data in form of publication, while a subscriber registers its interest in form of a subscription to receive publications of interest [6][10]. In broker–based pub/sub systems, a dedicated overlay network formed by a set of brokers, is used to connect publishers and subscribers anonymously. Pub/sub is an active research area that attracts attention from industry [9, 12, 15, 16, 21, 29] and academia [2–4, 8, 18, 18–20, 24, 26, 28, 30]. In advertisement–based pub/sub systems, a publisher first broadcasts an advertisement message about the data it wants to publish. The advertisement is saved in Subscription Routing Table (SRT) at each broker to form an advertisement–tree rooted at the publisher. To form publication routing paths, a subscription is forwarded in reverse direction to brokers that forwarded advertisements matching that subscription. Subscriptions are saved in Publication Routing Table (PRT) [6, 10] (in rest of this paper, an advertisement–based pub/sub system is referred to as pub/sub).

Tight coupling indicates that an advertisement and matching subscription have to be saved in routing tables of the same broker to generate publication routing paths. A subscription has to be reissued when tree of the matching advertisement changes (or the advertisement is reissued) for some reason. Most pub/sub systems use acyclic topologies which provide a single routing path between a pair of publisher and subscriber. This offers limited flexibility to deal with network condition like link congestion. Multiple paths may be available in a cyclic overlay, only one path at a time can be used for content–based routing with almost no support for dynamic routing [6]. We argue that tight coupling requires instantaneous updates in routing tables to generate new routing paths to avoid congested links in cyclic overlays [17]. Updates in routing tables generate extra traffic, cause message loss, and impede in–broker processing thus unsuitable for large networks and delay sensitive applications. Pub/sub systems that use cyclic overlays face additional challenges. (i) API calls create a large number of updates in routing tables because there are no covering techniques available for cyclic overlays — subscribe or unsubscribe calls require adding or removing the corresponding subscription in all PRTs, which may cause severe congestion [5, 7]. The worst impact is expected when a publisher calls unadvertise and not only the corresponding advertisement has to be removed from all SRTs but subscription–trees of the matching subscriptions may also be removed or pruned. (ii) An advertisement has to be unique identified to avoid message loops. (iii) Advertisement broadcast generates extra Inter–broker Message (IMs) to detect and discard duplicates. (iv) Generating routing paths of minimum lengths is difficult, and to the best of our knowledge, never addressed before (cf. Sec. 2). Indeed, a new architecture is needed to address current research challenges in pub/sub systems that use cyclic overlay.

We introduce OctopiA, a distributed pub/sub system, which offers inter–cluster dynamic routing by eliminating tight coupling. OctopiA is based on a novel concept called Subscription Subgrouping (or Subgrouping), which creates disjoint subsets of subscriptions, where each subset is known as Subscription Subgroup (or Subgroup) (cf. Sec. 5). Subgrouping requires that each subgroup should have distinct subscriptions. For example, for subgroups S1 and S2, there exists no subscription s such that s ∈ S1 ∩ S2. To realize subgrouping, OctopiA leverages a new Structured Cyclic Overlay Topology (SCOT) to formulate clusters of brokers where the number of clusters is equal to the number of subgroups. A cluster–level subscription broadcast algorithm forwards subscriptions in a subgroup to brokers in a cluster. Advertisements are forwarded to a set of brokers, called a region, and create advertisement–trees of length 1 (these patterns of subscription and advertisement forwarding eliminate tight coupling). OctopiA does not assign identifications to advertisements and generates no extra IMs. Since subscriptions in a subgroup are confined to one cluster, subscribe or unsubscribe calls generate updates in routing tables of brokers of only one cluster without effecting network traffic in other clusters. Similarly, advertise or unadvertise generate updates in routing tables of only one region without effecting network traffic in other regions. This pattern of forwarding subscription and advertisement improves throughput of a pub/sub in face of concurrent churn events. As subscriptions are confined to only one cluster, inter–cluster publication routing may generate
false positives because some clusters may receive publications but have no interested subscribers. An added novelty in OctopiA is the use of a bit-vector to eliminate false positives.

This paper makes the following contributions. Sec. 2 provides details of background issues in pub/sub systems that use cyclic overlays. Sec. 3 provides a comprehensive overview of the Structured Cyclic Overlay Topology (SCOT) used by OctopiA. Additional structural properties induced in SCOT to offer dynamic routing are discussed in Sec. 4. The novel concept of Subscription Subgrouping is discussed in Sec. 5. Advertisement and subscription broadcast algorithms in clustered SCOT are discussed in Sec. 6 and 7, respectively. Sec. 8 presents our bit-vector (or Cluster Index Vector) mechanism to avoid false positives. Algorithms for static and inter-cluster dynamic routing of publications are discussed in Sec. 9. Evaluation and comparison with identification-based (or TID-based [22]) state-of-the-art is presented in Sec. 10.

2 BACKGROUND ISSUES

This section describes issues related to handling message loops or cycles in pub/sub systems.

Advertisement–tree Identification: Cycles generate duplicate advertisements, subscriptions, and publications and it is important to detect and discard them. In the advertisement broadcast, each broker saves the advertisement in the form of \( \langle \text{advertisement, lasthop} \rangle \) tuple where the lasthop indicates the last sender (publisher or broker). Fig. 1(a) shows that Broker 5 may receive the advertisement of P twice from brokers 4 and 2, which generates cycles [22]. To resolve this issue, the host brokers of publishers assign a Unique Identification (UI) to each advertisement, which is used by other brokers to detect duplicates [5, 17, 22]. In Fig. 1(a), broker 5 discards the advertisement of P when received from broker 2 (assuming that it has already received the advertisement from broker 4). UIs require extra in–broker processing, increase delivery delays and payload of messages [22].

Extra Inter–broker Messages (IMs): Similar to duplicates in the previous issue in Fig. 1(a), Broker 6 discards the duplicate when received from Broker 3. This indicates that despite using UIs, extra IMs still generate to detect duplicates. The number of these extra IMs could be 80% of the total network traffic generated in advertisement broadcast [22]. This considerably affects the network performance when churn rate of publishers is high.

Advertisement–tree Length: In an acyclic overlay, advertisement broadcast generates a unique advertisement–tree even if the advertisement is issued multiple times (after calling unadvertise from the same broker). Advertisement broadcast in cyclic overlays is an uncontrolled process, which selects the first available link (or broker) as the next destination. This may generate advertisement–trees of larger lengths when load on the links and brokers is uneven. Multiple advertisement calls from the same broker may generate multiple advertisement–trees with different lengths. To the best of our knowledge, no advertisement broadcast algorithm exists that guarantees advertisement–trees of shortest lengths. An overlay link in this paper is represented as \( l_{\langle \text{source}, \text{destination} \rangle} \), where the source identifies message sending broker and the destination represents the message receiving broker. As Fig. 1(b) shows that the length of the advertisement–tree of P is twice of the length in Fig. 1(a). This is due to uneven load on the overlay links \( l_{\langle 1, 4 \rangle} \) and \( l_{\langle 2, 5 \rangle} \) at the time P issued the advertisement. Broker 5 receives the advertisement from Broker 6 and discards a duplicate when received from Broker 2. Similarly, Broker 4 discards a duplicate received from Broker 1 (assuming that it has already received the same advertisement from Broker 5). Length of the publication routing path (or the subscription–tree) in this case is also larger than Fig. 1(a). Although P and S are hosted by brokers of \( R_1 \), each publication is processed by brokers in the other two regions to reach Broker 4. This indicates that control over length of the advertisement–tree can reduce publication delay. Hop count to generate shortest advertisement–trees is not a suitable method because it can generate large number of updates in subscription–trees if the advertisement with least hop count is received as duplicate. Ideally, an advertisement–tree should always have the shortest length, even if some links have high loads when the advertisement is issued.

Binding UI: A subscription is bounded to UI of a matching advertisement to avoid cycles in publications routing [22]. A subscription may be bounded to more than one UIs depending on the number of matching advertisements. A broker may receive multiple copies of the same subscription bounded with multiple UIs and coming from different paths. This particular case is explained in Fig. 1(c), where S is interested in publications from P1, and P2. The subscription of S, which is received twice by Broker 2 from the last hops (Broker 3 and Broker 5), bound to UIs of advertisements of P1 and P2. Each publication has to carry UI of the corresponding advertisement–tree for routing to the next destination [22].

Single Routing Path: Only one path activated by the advertisement and subscription is available for routing publications, which poses a serious challenge in offering dynamic routing. A naive approach would suggest generating alternative routing paths instantaneously, however, this requires a large number of updates in routing tables to alter advertisement– and subscription–trees [6, 17]. In Fig. 1(a), if the link \( l_{\langle 1, 4 \rangle} \) becomes unavailable, alternative routing paths can be created by removing the link \( l_{\langle 1, 4 \rangle} \) and adding the link \( l_{\langle 2, 5 \rangle} \) or \( l_{\langle 3, 6 \rangle} \) in the advertisement–tree. This requires calling unadvertise to eliminate the current advertisement–tree and then generating the desired advertisement–tree using some controlled advertisement broadcast algorithm which is complex for cyclic overlays. In parallel to this, S should be informed to unsubscribe and resubscribe to receive publications from P. The whole process of removing old and adding new links requires updates in SRTs and PRTs, which is inefficient for large network settings. Dynamic routing is also supported using overlapping advertisement–trees [22]. In Fig. 1(c), advertisement–trees of P1 and P2 overlap at Broker 2, which indicates that Broker 2 can select least loaded link from \( l_{\langle 2, 3 \rangle} \) and \( l_{\langle 2, 5 \rangle} \) to forward a publication from P1 or P2 (recall that subscription of S matches with advertisements of P1 and P2). However, this approach requires overlapping advertisement–trees, which are not always possible because subscribers and publishers are anonymous to each other. A dynamically selected path may not have more brokers with overlapping advertisement–trees. In the above example, only one broker can support dynamic routing while a number of brokers have to save the subscription twice [22].
3 STRUCTURED CYCLIC TOPOLOGY

In this section, we describe how Structured Cyclic Overlay Topologies (SCOT) are created using Cartesian Product of Undirected Graphs (CPUG) [27].

3.1 Preliminaries

A graph is an ordered pair \( G = (V_G, E_G) \), where \( V_G \) is a finite set of vertices and \( E_G \) is a set of edges or links that connect two vertices in \( G \). The number of vertices of \( G \) (called order) is \( |G| \) or \( |V_G| \). Similarly, the number of edges in \( G \) is \( |E_G| \). A graph in which each pair of vertices are connected by an edge is called a complete graph. The diameter of a graph \( G \), represented as \( \text{diam}(G) \), is the shortest path between the two most distant nodes in \( G \).

A graph product is a binary operation that takes two small graph operands—for example \( G V_G, E_G \) and \( H V_H, E_H \)—to produce a large graph whose vertex set is given by \( V_G V_H \). Many types of graph products exist, but we find the Cartesian product most suitable for content-based routing. Other products, for example, the Direct product and the Strong product can be used but their rule-based interconnection of vertices increases node degree and makes routing complex. The CPUG of two graphs \( G V_G, E_G \) and \( H V_H, E_H \) is denoted by \( G \Join H \), with vertex set \( V_{G \Join H} \) and set of edges \( E_{G \Join H} \). Two vertices \( g, h \in V_{G \Join H} \) and \( g', h' \in V_{G \Join H} \) are adjacent if \( g = g' \) and \( h h' \in E_{G \Join H} \) or \( g g' \in E_{G \Join H} \) and \( h = h' \). Formally, the sets of vertices and edges of a CPUG are given as [14].

\[
V_{G \Join H} = \{g, h | g \in V_G \land h \in V_H\} \quad (1)
\]

\[
E_{G \Join H} = \{(g, h g', h') | g = g', h h' \in E_H\} \quad (2)
\]

The operand graphs \( G \) and \( H \) are called factors of \( G \Join H \). CPUG is commutative—that is, \( G \Join H = H \Join G \). Although CPUG of a number of graphs is possible, we are concerned with CPUG of only two graphs.

3.2 Structured Cyclic Overlay Topology

The Structured Cyclic Overlay Topology (SCOT) is a CPUG of two graphs. One graph, represented by \( G_{af} \), is called SCOT acyclic factor, while the second graph operand, represented by \( G_{cf} \), is called SCOT connectivity factor. A SCOT has two important properties: (i) Acyclic Property emphasizes that the \( G_{af} \) must be an acyclic graph, and (ii) Connectivity Property requires that \( G_{cf} \) must be a complete graph. These properties augment a SCOT with essential characteristics that are used for generating advertisement–trees of shortest lengths and control message cycles efficiently. \( V_{af} \) and \( V_{cf} \) are the sets of vertices of \( G_{af} \) and \( G_{cf} \), while \( E_{af} \) and \( E_{cf} \) are the sets of edges of \( G_{af} \) and \( G_{cf} \), respectively. For a singleton graph of vertex set \( \{h\} \subset V_{cf} \), the graph \( G_{cf} \) generated by \( G_{af} \Join \{h\} \) is called a \( G_{af} \) – fiber with index \( h \). Similarly, for a singleton graph of vertex set \( \{m\} \subset V_{af} \), the graph \( G_{af} \) generated by \( \{m\} \Join G_{cf} \) is called a \( G_{cf} \) – fiber with index \( m \). For each vertex of \( G_{cf} \) (and \( G_{af} \) ) CPUG generates one replica of \( G_{af} \) (and \( G_{cf} \) ). The number of distinct fibers of \( G_{af} \) and \( G_{cf} \) is equal to \( |V_{af}| \) and \( |V_{cf}| \) respectively. We describe the importance of using indexes in SCOT fibers in Sec. 4.

In addition to acyclic and connectivity properties, a SCOT has two more properties: (i) Index property, which emphasizes that the labels of nodes of \( G_{cf} \) must be a sequence of unique integers starting from zero, and (ii) Label Order property, which requires that the first operand (from left to right) of a CPUG node should be from node of \( G_{af} \). The index property implies that the index of each fiber of \( G_{af} \) is always an integer. The label order property indicates that the first part of the label of a SCOT node comes from the corresponding vertex of \( V_{af} \), and the second part is the label of the corresponding vertex of \( V_{cf} \), as indicated by Eq. 1. Reversing the order of operands

---

Figure 1: A cyclic overlay of six brokers is divided into three geographical regions \( R_1, R_2 \) and \( R_3 \). Each region contains two brokers: \( R_1 \) contains brokers 1 and 4; \( R_2 \) has brokers 2 and 5; \( R_3 \) has brokers 3 and 6. In (a) and (b), the subscriber \( S \) is interested in receiving publications from the publisher \( P \). In (c), \( S \) is interested in receiving publications from the publishers \( P_1 \) and \( P_2 \).

---

Figure 2: Operands of CPUG: Left of \( \square \) is \( G_{af} \), which is an acyclic graph; right of \( \square \) is \( G_{cf} \), which is a complete graph (a triangle).

---

Figure 3: Structured Cyclic Overlay Topology (SCOT) generated by CPUG of the operands in Fig. 2.
does not generate extra links or nodes, since CPUG is commutative. These two properties are used for clustering and routing purposes (cf. Sec. 4–9). Fig. 2 shows operands of the SCOT shown in Fig. 3.

4 STRUCTUREDNESS

This section describes a set of classifications for brokers and links to induce a structuredness in SCOT. The structuredness divides a SCOT into uniquely identifiable clusters and regions, which are used in routing advertisements, subscriptions, and publications.

4.1 Cluster and Region

Each \( G_{af} \) — fiber in a SCOT is a separate group of brokers called a SCOT Cluster (or simply a cluster) and represented by \( C_i \), where \( i \in V_{cf} \) is known as Cluster Index. A cluster index is the label of a vertex of \( V_{cf} \) that generates the cluster (or \( G_{af} \) — fiber) when a CPUG is calculated. Similarly, each \( G_{cf} \) — fiber is called a Region and is represented as \( R_j \), where \( j \in V_{af} \) is a Region Index. A region index is the label of a vertex of \( V_{cf} \) that generates the region (or \( G_{cf} \) — fiber) when a CPUG is calculated. There are \( |V_{cf}| \) and \( |V_{af}| \) number of clusters and regions in a SCOT, respectively. SCOT in Fig. 3 contains three clusters (horizontal layers) each identified by \( C_i \), where \( i \in \{0, 1, 2\} \), and six regions (vertical layers) each identified by a unique \( R_j \), where \( j \in \{a, b, c, d, e, f\} \).

4.2 Classifications

We assign types to clusters, brokers, and links to classify them for ease in routing advertisements, subscriptions, and publications. A SCOT has two types of overlay links: (i) an intra–cluster overlay link (aLink), and (ii) an inter–cluster overlay link (iLink). aLinks connect brokers in the same cluster, while iLinks connect brokers in the same region (Fig. 3). Messaging onto aLinks and iLinks is referred to as intra–cluster and inter–cluster messaging respectively. There are two types of clusters in a SCOT: (i) a Primary or Host Cluster, and (ii) a Secondary Cluster. A cluster that contains the host broker of a client (publisher or subscriber) is the host or primary cluster, while all other clusters are the secondary clusters of the client. The Primary neighbours of a broker are its direct neighbours in the same cluster, while the Secondary neighbours (or Secondary brokers) are those in the same region. Host region of a client is the region that contains host broker of the client. Host cluster of a publisher is its Primary Target Cluster (or Target Cluster) if at least one matching subscription is available in that cluster. An edge broker has at most one primary neighbour, while an inner broker has at least two primary neighbours. All brokers in a region are the same type (i.e., are either inner or edge). A SCOT broker in this paper is represented as \( B(x, y) \), where \( x \in V_{af} \) and \( y \in V_{cf} \). In Fig. 4, \( C_{0} \) is the host cluster, while \( C_{1} \) and \( C_{2} \) are secondary clusters of \( P_{0} \). Host brokers of \( P_{1} \) (i.e., \( B(e,1) \)), and \( P_{2} \) (i.e., \( B(a,2) \)) are inner and edge brokers respectively.

5 SUBSCRIPTION SUBGROUPING

Subscription Subgrouping (or Subgrouping) divides subscriptions into multiple disjoint sets, where each subset is known as Subscription Subgroup (or Subgroup). Subgrouping requires that each subgroup should have distinct subscriptions, that is, for subgroups \( S_{1}, S_{2}, S_{3}, \ldots, S_{n}; S_{1} \cap S_{2} \cap S_{3} \cap \ldots \cap S_{n} = \emptyset \). Subscriptions in a subgroup are available in the same exclusive overlay network area. This provides multiple benefits: (i) updates in routing tables due to subscribe or unsubscribe calls effect a limited area of the network, (ii) performance disruption due to any reason can be restricted to a limited network area, (iii) support for community clustering [23], and (iv) locality–based QoS provisioning [1]. Importantly, subgrouping could be exploited to eliminate tight coupling to offer dynamic routing without making updates in routing tables. OctopiA uses clustered SCOT to realize subgrouping by adopting specific patterns of subscription and advertisement broadcast (Secs. 6 & 7). The cluster–level subscription broadcast limits the availability of a subscription at a cluster. As envisioned in [1], QoS management and provisioning in large publish/subscribe systems should be based on locality considerations and could be used to determine partitions of a large overlay network into small logical units, each managing its own independent QoS control and provisioning to have multi–level QoS management. Clusters in a SCOT can be deployed in different geographical locations to connect local and remote publishers with subscribers. Each cluster (and subgroup) can have its own QoS provisioning, thus offering cluster–level demand–based multiple QoS at the same time. Finally, social networking systems like Facebook and Twitter can benefit from subgrouping to adopt community clustering, which is known to improve throughput of a pub/sub system by connecting local publishers and subscribers [23]. This paper focuses on the use of subgrouping to eliminate tight coupling to offer dynamic routing when congestion is detected.

6 ADVERTISEMENT BROADCAST

In OctopiA, the Advertisement Broadcast Process (ABP) forwards an advertisement to secondary brokers in the host region of the publisher, while the secondary brokers do not forward the advertisement any further. An advertisement is saved in Cluster Link Table (or CLT) to create an advertisement–tree of length 1, which is used for inter–cluster routing of publications (cf. Sec. 9). In Fig. 4, the advertisements issued by \( P_{0} \), \( P_{1} \), and \( P_{2} \) are broadcast to brokers in their respective host regions, which makes each advertisement available at most one broker of each cluster. In parallel to help in eliminating tight coupling, this approach has some additional benefits: (i) size of routing table reduced dramatically, (ii) updates in CLT generate only in host region of a publisher when advertise or unadvertise are called, and (iii) an unadvertise call does not require any possible unsubscribe calls. OctopiA assigns no identification to advertisement–trees and generates no IMs to detect and discard duplicates. When an unadvertise call is received, OctopiA simply
removes the advertisement from CLTs of brokers in the host region.

Advertisements forwarding by three publisher $P_0$, $P_1$, and $P_2$. The host regions of $P_0$, and $P_2$ is $R_{ad}$, and for $P_1$ is $R_c$.

7 SUBSCRIPTION BROADCAST

Cluster–level subscription broadcast of subscribers $S$, $S_0$, $S_1$, and $S_2$. The host cluster of $S$ and $S_0$ is $C_0$, while for $S_1$, and $S_2$ are $C_1$, and $C_2$ respectively.

To realize subgrouping and eliminate tight coupling, the Subscription Broadcast Process (SBP) has to make sure that the subscriptions in a subgroup are confined to a single cluster. Therefore, SBP forwards a subscription to brokers in the host cluster of the subscriber, which is similar to subscription broadcast in traditional acyclic overlays [6]. As shown in Fig. 5, subscription of $S_0$ is confined to brokers of its host cluster $C_0$ to form the subscription–tree. Similarly, subscription–trees of $S_1$ and $S_2$ are formed in their host clusters. SBP does not use trees of matching advertisements to generate publications routing paths [6]. Instead, subscription– and advertisement–trees are used independently for intra– and inter–cluster routing, respectively. When a subscriber calls unsubscribe, the corresponding subscription is simply removed from PRTs of brokers in the host cluster. This does not generate traffic in secondary clusters, which is even more effective when churn rate of subscribers is high.

8 CLUSTER INDEX VECTOR

A publisher may have interested subscribers in multiple clusters. As subscriptions in a subgroup are broadcast to only one cluster (and not available to other clusters), knowing the presence of interested subscribers in multiple clusters to send them publications is an issue. A naive solution suggests that publications should be forwarded on each iLink of the host broker of a publisher. However, this generates a number of false positives.

OctopiA uses a novel concept of Cluster Index Vector (CIV) to identify TSCs without forwarding subscriptions to other (non-host) clusters. CIV is a row (vector) of bits which identifies the host broker of a client, and the TSCs of a publisher. It is used to avoid false positives in static and dynamic routing of publications. Bits in CIV are indexed from right to left, with the index of the rightmost bit being zero (Fig. 6). Each bit of CIV, called the (cluster) index bit, is reserved for a cluster where index of the bit is same as index of the cluster it represents. Index bit of a broker is same as the index bit of the cluster it belongs to, while index of a client is the index of its host broker (or host cluster). Based on semantics, CIV has three contexts: (i) advertisement context ($CIV_a^i$), (ii) subscription context ($CIV_s^j$), and (iii) publication context ($CIV_p^k$). $i \in V_C$ is the index of the host cluster, while $j \in V_f$ is the index of the secondary cluster of a publisher. $CIV_a^i$ and $CIV_s^j$ are associated with advertisements. More specifically, $CIV_a^i$ is the context of the advertisement at the host broker of the publisher, and changes to $CIV_f$ when the advertisement reaches secondary brokers. $CIV_p^k$ is used in dynamic routing of publications. Although maximum number of meaningful bits in CIV are equal to the number of SCOT clusters, only two index bits in $CIV_f$ are meaningful. (i) $CIV_f$ identifies index of the advertisement sending cluster $C_j$, and (ii) $CIV_f^j$ is used by the advertisement receiving secondary cluster $C_j$. In Fig. 6(a), $CIV_a^0$ equal to 1 indicates that the publisher is connected with some broker of cluster $C_1$. In Fig. 1(b), $CIV_f^0$ equal to 1 indicates that $C_1$ is the host cluster of the advertisement issuing publisher, while $CIV_f^0$ shows that $C_2$ is the secondary cluster that received the advertisement. $CIV_p^0$ in Fig. 1(c) illustrates that the publisher is hosted by a broker of $C_1$, where $CIV_p^0$ has to be zero to avoid message loops in dynamic routing (Sec. 9). In the following, we discuss how index bits in CIV are set and reset to avoid false positives.

CIV for Advertisement — After receiving an advertisement, the host broker of a publisher in its host cluster $C_j$ performs four tasks: (i) creates $CIV_a^i$, (ii) sets $CIV_s^j$ to 1 and all other index bits to 0, (iii) saves the advertisement in Cluster Link Table (CLT) in the form of $\langle \text{advertisement}, CIV_{ad}^i, \text{lasthop} \rangle$ tuple, and (iv) forwards the advertisement with $CIV_s^j$ to secondary brokers. After receiving the advertisement, a secondary broker in a secondary cluster $C_j$ performs two tasks: (i) sets $CIV_f^j$ to 1 if one or more matching subscriptions are found at cluster $C_j$, (ii) saves the advertisement in CLT in form
of \{advertisement, CIV\}_j, lasthop\} tuple. When \(\text{CIV}\_j\) is set to 1, a \(\text{CIV} \_\text{SET}\) message is sent to the host broker of the publisher in 
\(C_i\) to indicate interested subscribers in \(C_j\). The host broker sets the
\(\text{CIV}\_j\) to 1 to mark that \(C_j\) is a TSC of the publisher. In Fig. 8, when \(B(e, 1)\) receives an advertisement from \(P_1\), it creates \(\text{CIV}\_j\), sets \(\text{CIV}\_j\) to 1 to all other bits to 0, and forwards the advertisement
(with \(\text{CIV}\_j\)) to secondary brokers \(B(e, 0)\) and \(B(e, 2)\). Assuming no matching subscriptions are found at clusters \(C_0\) and \(C_2\), both \(\text{CIV}\_0\) and \(\text{CIV}\_2\) are set to 0. In this scenario, \(\text{CIV}\_0\), \(\text{CIV}\_0\), and \(\text{CIV}\_2\) are 010, which indicates that \(P_1\) has no TSC. For \(P_2\), we assume that cluster \(C_0\) hosts interested subscribers. Secondary broker \(B(a, 0)\) sets the
\(\text{CIV}\_0\) to 1 and sends a \(\text{CIV} \_\text{SET}\) message to \(B(a, 2)\) to indicate that 
\(C_0\) is a TSC of \(P_2\). Upon receiving the \(\text{CIV} \_\text{SET}\) message, \(B(a, 2)\) sets \(\text{CIV}\_2\) to 0. \(\text{CIV}\_2\) for the advertisement issued by \(P_2\) is 101. Algorithm 1 shows that the host broker of the publisher sets the
bits and forwards \(\text{CIV}\_0\) and the advertisement to secondary brokers
(lines 2–8). Each secondary broker sends only one
\(\text{CIV}\_0\) bits and forwards the
advertisement, \(\text{CIV}\_i\), which indicates that
clusters \(C_0\) and \(C_2\) are TSCs of the publisher hosted by a broker of
cluster \(C_1\). Similarly, Fig. 8(b) shows that the advertisement is received from a broker of cluster \(C_2\) by a publisher hosted by a broker
of cluster \(C_1\), while \(C_2\) host one or more interested subscribers as
\(\text{CIV}\_2\) is 1. A \(\text{CIV} \_\text{UNSET}\) message is sent to the host broker
of a publisher when the last interested subscriber in a TSC \(C_j\) calls 
\text{unsubscribe}. After receiving the \(\text{CIV} \_\text{UNSET}\) message, the index
bit \(\text{CIV}\_j\) is set to 0 to avoid false positives.
Algorithm 2 indicates that a subscription is broadcast to brokers
in its host cluster (lines 1-3). \(\text{getDistinctMatchingAdvss}\) returns at
most one matching advertisement per iLink for which \(\text{CIV}\_j\) is 0.
\(\text{CIV} \_\text{SET}\) message is sent to each secondary broker to set \(\text{CIV}\_j\) to 1
(lines 6-10).

Algorithm 1: \text{scotAFPa}

\textbf{Input:} \(a\) : an advertisement message; 
\textbf{Output:} \(DL\) : a list of next destinations that should receive \(a\);

\begin{algorithmic}[1]
\State /* set the civ-bits at the host broker of the publisher */
\If {isHostBroker} 
\State \(\text{CIV}\_j\leftarrow 1;\)
\State hostBroker \leftarrow false;
\For {\(n \in \text{SecondaryNeighbours}\)}
\State a.next \leftarrow n;
\EndFor
\State DL.adda;
\State CLT.inserta;
\Else 
\If {a.type \(= \text{CIV} \_\text{SET}\)}
\State savedAdv \leftarrow getAdv:a:UID;
\State \(\text{CIV}\_j\leftarrow a: \text{CIV}\_j;\)
\EndIf
\Else 
\State /* send \text{CIV} \_\text{SET}\) message if matching subscription is found */
\If {isMatchingSubs = true}
\State a.next \leftarrow a.last;
\State a.CIV \leftarrow \text{CIV} \_\text{SET};
\State a.CIV \_j\leftarrow 1;
\State DL.adda;
\EndIf
\State CLT.inserta;
\EndIf
\EndIf
\EndFor
\end{algorithmic}

Algorithm 2: \text{scotSBPs}

\textbf{Input:} \(x\) : a subscription message; 
\textbf{Output:} \(DL\) : a list of next destinations that should receive \(x\);

\begin{algorithmic}[1]
\State foreach \(n \in \text{PrimaryNeighbours} \rightarrow \text{Sender}\)
\State x.next \leftarrow n;
\State DL.addx;
\State PRT.insertx;
\State /* select distinct advertisements which have \text{CIV}\_j = 0 */
\State \(\text{advxs} \leftarrow \text{getDistinctMatchingAdvss}\);
\For {\(a \in \text{advxs}\)}
\State \(\text{CIV} \_\text{SET}\) next \leftarrow a.last;
\State \(\text{CIV} \_\text{SET}\) sub \leftarrow x;
\State DL.addx:CIV\_SET;
\EndFor
\end{algorithmic}

9 PUBLICATIONS ROUTING

9.1 Static Routing

Static Publication Routing (SPR) algorithm uses advertisement–trees of length 1 and cluster–level subscription–trees for publication routing in a clustered SCOT. The algorithm deals with two scenarios. (i) When the host cluster of a publisher is the only target cluster, and lengths of the routing paths satisfy the relation:
\[ \max(d(u_1, v_1, u_2, v_2)) \leq \text{diam}G_{af} \]
where \(u_1, v_1\), and \(u_2, v_2\) are any brokers in the same cluster. No inter–cluster messaging takes place as no publication is forwarded to secondary brokers because all \(\text{CIV}\_j\), for secondary clusters, are 0. (ii) When at least one interested subscriber is hosted by a secondary cluster and therefore lengths of the routing paths satisfy the relation:
\[ \max(d(x_1, y_1, x_2, y_2)) \leq \left(\text{diam}G_{af} + 1\right) \]
where \(x_1, y_1\), and \(x_2, y_2\) are any brokers in a clustered SCOT. The host broker of a publisher forwards publications onto target iLinks (recall that a target link is part of a routing path to next destination)
A SCOT overlay of 12 brokers (4 clusters, 3 regions). The set of subscribers \( \{S_1, S_2, S_3, S_4\} \) is interested in publications from a publisher \( P \) hosted by the broker \( B(b,3) \).

for which index bits \( CIV_j \), where \( j \in V_c \), are 1. This avoids false positives by sending publications to those clusters which have interested subscribers. After the first broker of a TSC receives the publications, only intra–cluster routing is used to forward the publications to interested subscribers in a TSC. No loops appear and no path identifications are required because each cluster is an acyclic overlay. In Fig. 8, \( CIV_d \) for advertisements from \( P_0, P_1, \) and \( P_2 \) are 110, 111, and 100, respectively. For \( P_0 \) and \( P_2 \), routing is only inter– and intra–cluster, respectively. Publications from \( P_1 \) result in both intra– and inter–cluster routing as the publisher has interested subscribers in all three clusters. \( B(d,1) \) forwards publications to primary broker \( B(d,1) \) therefore messaging onto the aLink \( l(e,1,d,1) \) is intra–cluster, while publications routing to secondary brokers \( B(e,0) \) and \( B(e,2) \) onto the iLinks \( l(e,1,e,0) \) and \( l(e,1,e,2) \) is inter–cluster.

![Diagram of SCOT overlay](image)

**Algorithm 3: scotSPRn**

**Input:** \( n \) : a publication message;  
**Output:** \( DL \) : a list of next destinations that should receive \( n \);

1. \( IS \leftarrow \) getDistinctSubs;  
2. \( *^! \) intra–cluster routing of \( n \) in host cluster. \( *^! \);  
3. \( IS \leftarrow IS \);  
4. \( DL \leftarrow DL \);  
5. \( CIV_d \leftarrow getCIV;n \);  
6. \( *^! \) inter–cluster routing of \( n \) to TSCs. \( *^! \);  
7. \( IS \leftarrow IS \);  
8. \( DL \leftarrow DL \);  
9. \( DL \leftarrow DL \);  
10. \( IS \leftarrow IS \);  
11. \( DL \leftarrow DL \);  
12. \( IS \leftarrow IS \);  
13. \( IS \leftarrow IS \);  
14. \( DL \leftarrow DL \);  
15. \( DL \leftarrow DL \);  
16. \( IS \leftarrow IS \);  
17. \( DL \leftarrow DL \);  
18. \( IS \leftarrow IS \);

9.2 Inter–cluster Dynamic Routing

Dynamic Routing refers to the capability of a pub/sub system to alter routing path in response to congested or failed links and/or congested or failed brokers. Multiple techniques have been proposed to handle dynamic routing in address–based networks where routing paths are calculated from a global view of a network topology graph that is saved on every network router [25]. IP addresses may be used to form clusters in a network area where same network mask can be used which requires a single table entry. However, these techniques are not applicable in pub/sub systems because brokers are aware of only their direct neighbours and destinations addressing is based on contents (i.e., subscriptions). Therefore dynamic routing decisions are decentralized and have to be made without having a global view of an overlay. This section describes Inter–cluster Dynamic Routing (IDR) algorithm. It deals with routing when one or more iLinks are congested and publications start queuing up, while discussion about dynamic routing when one or more brokers fail is out of scope of this paper.

OctopiA leverage the structuredness of SCOT to realize sub grouping and eliminate tight coupling to achieve inter–cluster dynamic routing when one or more iLinks are congested. Our approach is scalable because it does not require updates in routing tables and can reduce delivery delays when a large number of publications start.
accumulating in the output queues. SPR algorithm adds exclusive copies of a publication in the output queues of the target links. If a publisher generates γ number of publications in t_w interval, and there are α number of target aLinks and β target iLinks, the host broker of the publisher enqueue α + β γ number of copies of a publication in t_w interval. A High Rate Publisher (HRP) with a high value of γ can overwhelm brokers when SPR algorithm is used. IDR algorithm alleviates overwhelmed brokers and dynamically tries to find uncongested iLinks to forward publications to TSCs. As parallel iLinks are available in a clustered SCOT, one can be selected without making updates in routing tables. The selected iLink may not be part of a subscription–tree to the TSC. IDR algorithm adds at most one copy of a publication, when there are multiple congested output queues of the target links at a broker. For the remaining TSCs, which are not forwarded the publication due to congested output queues, the algorithm sets their index bits in CIV_p to 1. CIV_p is then added in the header of the publication that is added in the output queue. The publication that carries CIV_p is identified as CIV_p Dynamic or CIV_p D. Using this technique, IDR algorithm keeps the length of the congested output queues of an overwhelmed broker to a minimum, and the load of forwarding the publication to the remaining TSCs, with help of their index bits in CIV_p, is shifted to other brokers using the heuristic that uncongested target iLinks are available down the routing path. OctopiA uses Eq. 3 to find whether an output queue is congested and IDR algorithm should be activated.

\[
Q_{in} + Q_{out} > \tau
\]  

Q_{in} and Q_{out} are the number of publications that enter into or leave the output queue in the time window t_w, respectively. The term 1 + Q_{in} Q_{out} is the ratio of 1 + Q_{in} to 1 + Q_{out}, and is known as the Congestion Element (CE). CE > 1 indicates that the congestion is increased, while CE < 1 shows that congestion is decreased in the last t_w interval. An output queue is congestion–free when CE is 1 and the queue length Q_l is 0. OctopiA saves the values of Q_{in} and Q_{out} in a Link Status Table (LST) on each broker, and the values are updated after each t_w interval. Dynamic routing with IDR is further explained with the help of three selected cases in the following.

**Case I – Overloaded iLinks:** This case describes dynamic routing when one or more (but not all) target iLinks are congested. The relation: \(1 \leq |iLink|_{OL} < \beta\), where |iLink|_{OL} is the number of congested target iLinks at a broker, expresses that one or more target iLinks are congested. After detecting the congested output queues, dynamic routing is activated by forwarding CIV_p D, where i ∈ V_{cf} identifies index of the broker (or cluster) which detected congestion, to the other brokers down the dynamic routing path. No congested output queue of the target iLinks receives the publication. CIV_p is added (with CIV_p D) into the output queue of an uncongested target iLink, and unlike to other two contexts, is not saved in any routing tables. If there are multiple uncongested target iLinks, CIV_p D is added into the output queue with the least value of Q_l. The number of publications added in t_w interval is α + β γ, where β is the number of uncongested target iLinks, and Q_l < 0. Fig. 7(b) shows that after detecting the congestion using Eq. 3, B(b, 3) switches to IDR algorithm with the two candidate iLinks, 1(b, 3, b, 2), and 1(b, 3, b, 1), that can receive CIV_3 D. Presumably, having the least value of Q_l, the link 1(b, 3, b, 1) is selected to route CIV_3 D. B(b, 3) sets the bit CIV_3 to 1 (and all other bits to 0), and adds CIV_3 D in the output queue of 1(b, 3, b, 1). The bit pattern 0001 of CIV_3 indicates that C0 is a TSC. As the target iLink 1(b, 1, b, 0) is also congested, the CIV_3 D is forwarded to B(c, 1) using the subscription–tree of S2. At B(c, 1), the publication is eventually forwarded to broker B(c, 0) of C0 as the target iLink 1(c, 1, c, 0) is uncongested. Since C0 was the only TSC, B(c, 1) removes CIV_3 from CIV_3 D and routing switches to static mode. To forward a publication to all subscribers, SPR algorithm generates 8 IMs, while IDR algorithm generated 7 IMs. The dynamic routing path generated by IDR algorithm for S1 contained one extra broker but no congested iLink.

**Case II – All iLinks Overloaded:** The condition \(|iLink|_{OL} = \beta\) at a broker indicates that all the target iLinks are congested. We assume that at least one target aLink is uncongested and can be selected to forward CIV_p D. If more than one aLinks are available, the one with least value of Q_l is selected. The number of publications added to the output queues is a γ. Fig. 7(c) illustrates that \(1(b, 3, b, 0)\) and \(1(b, 3, b, 2)\) are congested and alternative paths to forward the publication to TSCs C0 and C2 should be searched. The link \(1(b, 3, b, 1)\) is not useful because it forwards publications to a non–TSC cluster C1 (Case III indicates that this generates loops). Presumably, having least value of Q_1, CIV_3 D (with CIV_3 0010) is added into the output queue of \(1(b, 3, c, 3)\). B(c, 3) forwards the publication onto two uncongested target iLinks to TSCs C0 and C2 after removing CIV_3. IDR generated 6 IMs and eliminated two congested target iLinks when sending a publication to S1, S2, and S5, while SPR algorithm generates 7 IMs.

**Case III – All Links Overloaded:** The condition \(|iLinks|_{OL} + |aLinks|_{OL} = \alpha + \beta\), where |aLinks|_{OL} is the number of congested target aLinks at a broker, indicates that no uncongested target link is available. CIV_p D is added into the congested output queue of the target iLink with least value of Q_l. The congested target aLink, even if has the least value of Q_l, is not selected because of the possibility of having more congested aLinks down the routing path in the same cluster. The 1 + α number of copies of the publication in generated in t_w interval. Fig. 7(d) shows that CIV_p D (CIV_3 0011) is forwarded onto the target iLink 1(b, 3, b, 2), which presumably has the least value of Q_l. At B(b, 2), the target link \(1(b, 2, b, 1)\) is also congested, while the target iLink 1(b, 2, b, 0) is uncongested. B(b, 2) sets CIV_3 0 to 0 and forwards CIV_3 D (with CIV_3 0010) to B(b, 0). CIV_p D indicates that the publication should be forwarded to C1. However, B(b, 0) forwards CIV_3 D to B(a, 0) because the target iLink \(|iLink|_{OL} = \beta, B(a, 0)\) is congested and a search for uncongested target iLink is still possible at B(a, 0) because the link \(|iLink|_{OL} = \beta, B(a, 0)\) is part of the subscription–tree of S1, which is not sent the publication yet. At B(a, 0), the only available target iLink \(a\) a, (a, 1) is also congested and there is no more aLink available in C0 cluster (the publication cannot be sent back to B(b, 0) to avoid message loops). As a last option, B(a, 0) forwards the publication to Ba, 1 ∈ C1 after removing CIV_3 from the header. IDR algorithm generated 8 IMs and could eliminate only one congested iLinks, while SPR algorithm generates 7 IMs to forward a publication. Dynamic routing paths for S1 and S2 had 3 additional brokers. To prevent message loop, cluster index bit of a TSC in CIV_p is set to 0 after the publication is delivered to any broker of the TSC. For example, in order to avoid the congested
In Algorithm 4, the getDistinctSubs method follows the principle of downstream replication and all target aLinks (whether overloaded) receive a copy of publication \( n \) (lines 2-5). The host broker of a publisher initially adds exclusive copies of \( n \) for next destinations to TSCs and sets the cluster index bits for overloaded iLinks (lines 7-16). A broker checks for the unoverloaded target iLink to forward a copy of \( n \) to a TSC (lines 18-24). The method iLinkOverloaded takes current cluster index \( j \) and list of all target iLinks \( \beta \) to check if an iLink is overloaded. The method get_iLink takes index of a TSC to provide an iLink to a broker of that TSC. \( \mu_1 \) and \( \mu_2 \) represent least loaded aLink and iLink respectively. The method ad justCIV\(_p\) prepares the CIV\(_p\) – D publication by adding CIV\(_p\) in the header of a copy of \( n \). Three cases are handled based on overloaded and unoverloaded iLinks and aLinks (lines 27-40).

IDR algorithm has two limitations: (i) no support for intra-cluster dynamic routing, and (ii) a subscription-tree of an interested subscriber is required in a cluster to search uncongested target links. For example, in Fig. 7(d), the publication has to be forwarded onto congested links at \( B(b, 3) \) although \( l(b, 3, a, 3) \) is uncongested. Since \( l(b, 3, a, 3) \) is not included in subscription-trees of \( S4 \) and \( S3 \), it cannot be used. An extended version of IDR algorithm to handle these limitations is part of the future work.

10 EVALUATION

We implemented SPR and IDR algorithms in OctopiA, developed on top of an open source pub/sub tool PADRES [10]. For a comparison with state-of-the-art, we used TID-based routing algorithms [22].

Setup — Fig. 9 shows the factors of SCOT topology \( S_e \). We used for evaluation and comparison of SPR and IDR algorithms with TID-based static (TID-S) and TID-based dynamic (TID-D) routing algorithms. The \( \alpha, \beta \) factor of \( S_e \) is an acyclic topology of 14 brokers, with 4 inner (F, G, H and I) and 10 edge brokers (A, B, C, D, E, J, K, L, M and N). \( S_e \) had 16 inner and 40 edge brokers for a total of 54 brokers and was deployed on a cluster of 7 computing nodes connected through one dedicated switch. Each node had 2 Intel Xeon E5-2620 CPU with total 12 cores of 2.1 GHz each, and 64 GB RAM.

Stock datasets are commonly used to generate workloads for evaluations of pub/sub systems [11]. We used a dataset of 500 stock symbols from Yahoo Finance!, where each stock publication had 7 metrics — to demonstrate the effectiveness, experiments were run in an unclustered (for PADRES) and clustered (for OctopiA) \( S_e \). We used the following metrics.

Advertisement delay is the maximum time elapsed by an advertisement to reach every broker of a cyclic overlay. In OctopiA, advertisement delay is expected to be less because an advertisement is broadcast to brokers in a region and generates advertisement-tree of length 1. In traditional pub/sub systems, Subscription delay is the maximum time elapsed as a subscription is forwarded from the subscriber to brokers with matching advertisements. OctopiA performs cluster-level subscription broadcast, without considering matching advertisement. The study of difference between subscription delays in two cases is worthwhile.

Publication delay measures end-to-end latency of a publication.
delivery and is an important metric. Matching delay is the time taken by a broker to find matching subscriptions. As number of subscriptions saved at each broker in both tools is expected to vary, this metric provides useful information about in–broker processing.

The number of Inter–broker Messages (IMs) depends on the number of overlay hops in routing paths. OctopiA generates routing paths of shortest lengths, therefore, investigating the number of IMs generated in both cases is important.

Size of SRT and CLT provide the number of advertisements saved in routing tables in both cases. As both tools use different approaches to forward advertisements, knowing difference of these values provide useful insight.

Size of PRT measures the number of subscriptions saved in PRTs. As both the tools use different SBP algorithms, it is important to investigate the difference between average size of PRTs. Aggregation techniques like covering can be used to reduce size of CLTs, SRTs and PRTs. These techniques are developed for acyclic overlays and can be used at cluster-level in OctopiA. However, covering is not considered because PADRES does not support covering in cyclic overlays.

Results — The results presented in this section cover four important aspects of evaluation: (i) Publisher Scalability, (ii) Subscriber Scalability, (iii) End to End publication delivery, and (iv) Dynamic Routing.

Publisher Scalability: We increased the number of publishers from 100 to 500 to measure impact of increasing advertisements in the systems. Fig. 7 shows that the average advertisement delay in unclustered $S_e$ (legend PADRES) was 266% higher than the clustered $S_e$ (legend OctopiA) and remained nearly constant. The reasons of this difference are larger lengths of advertisement–trees and extra IMs generated to detect and discard duplicates by PADRES. OctopiA generates advertisements–trees of length 1, and extra IMs are not required in ABP. Fig. 8 shows the average size of CLTs (in OctopiA) was reduced by 93% when compared with SRTs (in PADRES). PADRES broadcasts an advertisement to all brokers in an overlay, while OctopiA forwards an advertisement to the secondary brokers in the host region of a publisher, which dramatically reduced size of CLTs. Fig. 9 shows that the number of IMs generated by ABP in PADRES was 88.5 times more than OctopiA. There are two reasons of this large difference: (i) PADRES broadcast advertisement in whole overlay, while OctopiA forwards advertisement only to brokers in the host region of the publisher, and (ii) PADRES generates a large number of extra IMs (upto 80% of the total traffic [22]) to detect and discard duplicates, while ABP in OctopiA does not generate extra IMs.

Subscriber Scalability: We increased the number of subscribers from 1000 to 10000, with 100 publishers, to study impact of increasing subscriptions. Fig. 11 shows that the average size of PRTs in PADRES was 33% smaller than OctopiA. PADRES forwards subscriptions in the reverse direction of the matching advertisement–trees, while OctopiA saves more subscriptions due to cluster–level subscription broadcast. This is the reason that OctopiA generated 32.5% more IMs than PADRES in SBP (Fig. 12). Although OctopiA saved more subscriptions and generated more IMs in SBP, Fig. 10 shows that subscription delay is OctopiA was 34% less than PADRES. The reason of this difference is the less matching delay due to smaller size of CLTs than SRTs (Fig. 8).

Publication Delivery: This part compares static and dynamic routing by forcing IDR to use static routing paths. Using this approach, we computed performance penalty due to selecting dynamic routing paths in IDR. Fig. 13 indicates that the average publication delay in OctopiA was 25% less than PADRES. The reason of this difference is the additional payload due to carrying TIDs with each publication onto longer routing paths generated by PADRES. There was a minimal difference in the publication delivery delay in static and dynamic routing in both cases. The collected data indicated that the publication delay by SPR algorithm was 0.8% less than IDR algorithm, while TID–S was 0.6% less than TID–D algorithm. This slight improvement with SPR and TID–S algorithms is due to the fact that static routing does not do additional checks to find alternative routing paths. The average matching delay in OctopiA was 11% higher than PADRES (Fig. 14). PADRES performs matching at the host brokers of interested subscribers, while OctopiA does this at every broker in a routing path. Fig. 14 also illustrates that the difference in matching delay decreased when the number of subscribers increased, which decreased the number of forward–only brokers. Although OctopiA generated more IMs in SBP with larger size of PRTs (Figs. 11, 12), Fig. 15 shows that OctopiA generated 4.5% less IMs than PADRES in publications routing. This indicates that OctopiA generated shorter publication routing paths in clustered $S_e$. The difference in IMs diminished with the increase in the number of subscribers, due to decrease in forward–only brokers.

Dynamic Routing: We now discuss dynamic routing and analyse stability of OctopiA when publications start queueing up due to a bursty publisher. The stability analysis demonstrate our approach by showing how quickly OctopiA converges to a normal state after an HRP finishes a burst. To activate dynamic routing, we used Eq. 3 in PADRES and OctopiA for a fair comparison. Eq. 3 has great similarity with the link utilization ratio that PADRES uses [22]. We set the value of $\tau$ to 10 and $t_o$ to 50 milliseconds for 10000 subscribers and 100 publishers. Each publisher generated 1000 publications with varying rate of 100 to 120 publications per minute. We also evaluated static routing using SPR and TID–S with the same workload. To support dynamic routing, PADRES requires that a subscription should match with multiple advertisements to increase possibility of finding alternative routing paths [22]. To handle this limitation, we increased selectivity of each subscriber from 2% to 4% to receive publications from 4 different publishers, however, availability of 4 alternative routing paths is not guaranteed as publishers and subscribers are anonymous. To exert more load on iLinks and aLinks, all clusters of $S_e$ were the target clusters of the HRP with 0.1% matching subscriptions. No subscriber (for HRP) was hosted by the host broker of the HRP. We used varying rate of the HRP to study impact of shorter and longer bursts. The HRP sent 50K publications at rates of 100K, 80K, 60K, and 40K publications per minute. We ran each simulation 20 times, for 80 times in total. The burst continued for 30 to 75 seconds depending on the burst rate. Each point in Figs. 17–20 is a maximum delivery delay of 5000 publications received in a sequence. This approach helps in analysing the stability without graphing too many points. On average, IDR algorithm stabilized OctopiA 35 seconds before SPR algorithm and generated only 0.04% more IMs. In the first 157 seconds, the maximum publication delay was the same in the
two algorithms and no tendency toward stability was observed. This indicates that, due to the high rates of publications, the state of the system (OctopiA) did not converge to normal until the condition $CE < 1$ was maintained for some time (on average 60 seconds for the four simulations). The average value of $Q_t$ of target links at the brokers in routing paths when IDR algorithm was used is 32% less than SPR algorithm. Thanks to the bit–vector approach, which uses $CIV_j$ for dynamic routing to improve performance of OctopiA by adding minimum copies of publications in the congested output queues. The analysis of the collected data indicates that 51% publications in IDR algorithm and 48% in SPR algorithm had delay less than one second. This indicates that IDR algorithm not just stabilized the system before SPR algorithm but also decreased the average delays. Figs. 17–20 indicate that static and dynamic routing in PADRES has varied performance. In some cases, static routing performed better than dynamic (Figs. 17, 20), while in other dynamic routing performed better than static routing (Fig. 18) or both had same performance (Fig. 19). For 80 simulations, TID–S performed better than TID–D in 65%, TID–D performed better than TID–S in 25%, while performance was same in 10% simulations. On average, publication delay by TID–D algorithm is 18% higher than the TID–S algorithm, which indicates that dynamic routing by PADRES when selectivity is 4% is poor than static routing. Average publication delay by OctopiA is 54% less than PADRES, which indicates an excellent improvement in burst scenario. There are multiple reasons for better performance. First, PADRES maintains subscriptions and TIDs in separate space for algorithmic reasons. When routing switches to dynamic, after finding matching subscriptions, PADRES performs a sequential search to find TIDs of the matched subscriptions to search un congested next destination paths. As covering is not supported for cyclic overlays, linear search over a complete subscription space is expansive and dwarf improvements of dynamic routing by increasing path selection delay. As Fig. 16 illustrates, the path selection delay in TID–D algorithm is 1230% higher than IDR and 600% higher than TID–S algorithms. On the other side, path selection delay in IDR algorithm is just 7% higher than SPR algorithm. Thanks to the bit–vector approach of IDR algorithm that exploits structuredness of $S_e$ to find next destination clusters without a linear search over saved subscriptions. Second, PADRES does not assure dynamic routing as overlapping advertisement–trees for a subscription are not guaranteed even if multiple copies of the same subscription is recorded at brokers [22]. Further, PADRES does not guarantee of having multiple brokers with overlapping advertisement–trees in a dynamic routing path. One broker may have overlapping advertisement–trees for a subscription to support dynamic routing, while other brokers, in the same dynamic routing path, may not support dynamic routing due to the unavailability of overlapping advertisement–trees. As a result, static routing performed better than dynamic routing in most of the cases. OctopiA uses structuredness of SCOT to avoid these limitations in offering inter–cluster dynamic routing. Third, longer routing paths generated by PADRES in unclustered $S_e$ causes higher delivery delays. Fourth, payload of a publication in PADRES is higher due to carrying TIDs. On average, TID–S stabilized PADRES almost 15 seconds before TID–D routing algorithm, while SPR and IDR algorithms stabilized OctopiA 200 seconds before PADRES.

11 RELATED WORK

Although content–based routing in broker–based pub/sub systems has been focus of many research efforts [3, 4, 8, 28], only SIENA [6] and PADRES [10] support routing in cyclic overlays. SIENA uses latency–based distance–vector algorithms for generating routing
paths [5]. The system, however, does not generate advertisement–trees of shortest lengths and dynamic routing is not supported. Shuping et al [17] divides a cyclic overlay into clusters to apply content–based inter– and intra–cluster routing of publications. The approach embeds a list of destination brokers to deliver publications, which consumes network bandwidth inefficiently. A broker has to be aware of other brokers in a cluster, which increases topology maintenance cost. Dynamic routing requires updates in routing tables, which increases network traffic and delivery delays. Li et al [22] offers dynamic routing without making updates in routing tables and their approach is closer to our work. Each advertisement–tree is uniquely identified and a large number of IMs are generated in advertisement broadcast to detected duplicates. This technique relies on intersecting advertisement–trees and does not support dynamic routing for subscriptions matching with single advertisement. Intersecting advertisement–trees are not always possible even if a subscription matches with multiple advertisements. OctopiA neither generates redundant IMs in advertisement broadcast nor requires carrying unique identification to discard duplicates. Resultantly, publications do not carry extra payload to identify routing paths to avoid message loops. Thanks to the structuredness of SCOT overlay, which inherently supports subgrouping and eliminates tight coupling. OctopiA requires a minimal topology maintenance cost as SCOT brokers are decoupled and aware of their direct neighbours only. Our algorithms generate advertisement–trees of length 1 and offers inter–cluster dynamic routing without requiring updates in routing tables. All these properties make OctopiA scalable and suitable for large pub/sub systems.

12 CONCLUSION
This paper introduces the first pub/sub system, OctopiA, that offers inter–cluster dynamic routing of publications. Tight coupling in pub/sub systems is one of the main obstacle that requires updates in routing tables to generate alternative routing paths. OctopiA uses a novel concept of subgrouping, which divides subscriptions into exclusive disjoint subsets. Subgrouping is realized using the structuredness of a clustered SCOT overlay, where subscriptions in a subgroup are available in an exclusive host cluster through the cluster–level subscription broadcast. An advertisement–trees in clustered SCOT has length 1. These patterns of subscriptions and advertisements broadcast eliminate tight coupling to enable inter–cluster dynamic routing without making updates in routing tables. As subscriptions are available only in their host clusters, OctopiA uses a bit–vector to identify TSCs to avoid false positives in inter–cluster routing of publications. OctopiA does not generate extra IMs to detect and discard duplicate advertisements and requires no path identification to route publications. Evaluation with real word data indicates that SPR and and IDR algorithms scale well when compared with state–of–the–art TID–based static and dynamic routing. Support for intra–cluster dynamic routing is part of the future work.

REFERENCES

[1] P. Bellavista, A. Corradi, and A. Reale. Quality of service in wide scale publish–subscribe systems. IEEE Communications Surveys Tutorials, 16(3):1591–1616, Third 2014.

[2] César Cailas, Kaiwen Zhang, Bettina Kemme, Jörg Kiemle, and Hans-Arno Jacobsen. Publish/subscribe network designs for multiplayer games. In Proceedings of the 15th International Middleware Conference, Middleware ’14, pages 241–252, New York, NY, USA, 2014. ACM.

[3] Fengyun Cao and J.P. Singh. Efficient event routing in content–based publish–subscribe service networks. In INFOCOM 2004. Twenty–third Annual Joint Conference of the IEEE Computer and Communications Societies, volume 2, pages 929–940 vol.2, March 2004.

[4] Fengyun Cao and J.P. Singh. Medym: match–early and dynamic multicast for content–based publish–subscribe service networks. In Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on, pages 370–376, June 2005.

[5] Antonio Carzaniga. Architectures for an event notification service scalable to wide–area networks. PhD thesis, politecnico di milano. 1998.

[6] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide–area event notification service. ACM Trans. Comput. Syst., 19(3):332–383, August 2001.

[7] Mingwen Chen, Songlin Hu, Vinod Muthusamy, and Hans-Arno Jacobsen. Congestion Avoidance with Incremental Filter Aggregation in Content–Based Routing Networks. In IEEE 35th International Conference on Distributed Computing Systems (ICDCS), June 2015.

[8] G. Cugola, Elisabetta Di Nitto, and A. Fuggetta. The jeti event–based infrastructure and its application to the development of the opms wms. Software Engineering, IEEE Transactions on, 27(9):827–850, 2001.

[9] Cooper et al. Pufts. Yahoo!’s hosted data serving platform. Proc. VLDB Endow., 1(2):1277–1288, August 2008.

[10] Hans-Arno et al. The PADRES Publish/Subscribe System. In Principles and Applications of Distributed Event–Based Systems, pages 164–205. IGI Global, 2010.

[11] Navneet et al. Minimizing the Communication Cost of Aggregation in Publish/Subscribe Systems. In 35th IEEE International Conference on Distributed Computing Systems (ICDCS), 2015.

[12] Yogeshwar Sharma et al. Worshop: Reliable pub–sub to support geo–replicated internet services. In NSDI 15, pages 351–366, Oakland, CA, MA, May 2015. USENIX Association.

[13] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. ACM Comput. Surv., 35(2):114–131, June 2003.

[14] Richard Hammack, Wilfried Imrich, and Sandi Klavzar. Handbook of Product Graphs. CRC Press, Inc., 2nd edition, 2011.

[15] Microsoft Bing Pulse (http://pulse.bing.com/).

[16] Google Cloud Pub/Sub (https://cloud.google.com/pubsub/).

[17] Shuping Ji, Chunyang Ye, Jun Wei, and Hans-Arno Jacobsen. Merc: Match at edge and route intra–cluster for content–based publish/subscribe systems. In Proceedings of the 16th Annual Middleware Conference, pages 12–24, 2015.

[18] Reza Sherafat Kazemzadeh. Overlay Neighborhoods for Distributed Publish/Subscribe Systems. PhD thesis, University of Toronto, 2012.

[19] Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen. Partition–tolerant Publish/Subscribe Systems. In 30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011), pages 101–110, Madrid, Spain, October 2011. IEEE.

[20] R. S. Kazemzadeh and H.-A. Jacobsen. Publity+: A peer–assisted publish/subscribe service for timely dissemination of bulk content. In Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on, pages 345–354, June 2012.

[21] Jay Kreps, Neha Narhked, and Jan Rao. Kafka: a distributed messaging system for log processing. NetDB Workshop, pages 01–07, 2011.

[22] Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsen. Adaptive content–based routing in general overlay topologies. Middleware ’08, pages 1–21, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[23] Wei Li, Songlin Hu, Jintao Li, and H.-A. Jacobsen. Community clustering for distributed publish/subscribe systems. In Cluster Computing (CLUSTER), 2012 IEEE International Conference on, pages 81–89, Sept 2012.

[24] H.A. Jacobsen R. Hull M. Sadoghi, M. Jergler and R. Vaculin. Safe distribution and parallel execution of data–centric workflows over publish/subscribe abstractions. IEEE Transaction on Knowledge and Data Engineering, 27(10):2824–2838, 2015.

[25] Deepankar Medhi and Karthikeyan Ramasamy. Network Routing: Algorithms, Protocols, and Architectures. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[26] Vinod Muthusamy. Flexible Distributed Business Process Management. PhD thesis, University of Toronto, 2011.

[27] E. Parsonage, H.X. Nguyen, R. Bowden, S. Knight, N. Fulkner, and M. Roughan. Generalized graph products for network design and analysis. In Network Protocols (ICNP). 2011 19th IEEE International Conference on, pages 79–88, Oct 2011.

[28] Helge Parzyjegla, Daniel Graff, Arnd Schr"oter, Jan Richling, and Gero M"uhl. Internet services. In Flexible Distributed Business Process Management, pages 164–205. IGI Global, 2011.

[29] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide–area event notification service. ACM Trans. Comput. Syst., 19(3):332–383, August 2001.
[29] Mohammad Sadoghi, Martin Labrecque, Harsh Singh, Warren Shum, and Hans-Arno Jacobsen. Efficient event processing through reconfigurable hardware for algorithmic trading. *Proc. VLDB Endow.*, 3(1-2):1525–1528, September 2010.

[30] Muhammad Adnan Tariq, Boris Koldehofe, Sukanya Bhowmik, and Kurt Rothermel. Pleroma: A sdn-based high performance publish/subscribe middleware. *Middleware ’14*, pages 217–228, 2014.