Content–based Dynamic Routing in Structured Overlay Networks

Muhammad Shafique
Alumni Carleton University Ottawa Canada
alumni@acm.org

ABSTRACT
Acyclic overlays used for broker–based publish/subscribe systems provide unique paths for content–based routing from a publisher to interested subscribers. Cyclic overlays may provide multiple paths, however, the subscription broadcast process generates one content–based routing path per subscription. This poses serious challenges in offering dynamic routing of notifications when congestion is detected because instantaneous updates in routing tables are required to generate alternative routing paths. This paper introduces the first subscription–based publish/subscribe system, OctopiS, which offers inter–cluster dynamic routing when congestion in the output queues is detected. OctopiS is based on a formally defined Structured Cyclic Overlay Topology (SCOT). SCOT is divided into homogeneous clusters where each cluster has equal number of brokers and connects to other clusters through multiple inter–cluster overlay links. These links are used to provide parallel routing paths between publishers and subscribers connected to brokers in different clusters. While aiming at deployment at data center networks, OctopiS generates subscription–trees of shortest lengths used by Static Notification Routing (SNR) algorithm. Dynamic Notification Routing (DNR) algorithm uses a bit–vector mechanism to exploit the structuredness of a clustered SCOT to offer inter–cluster dynamic routing without making updates in routing tables and minimizing load on overwhelmed brokers and congested links. Experiments on a cluster tested with real world data show that OctopiS is scalable and reduces the number of inter–broker messages in subscription delivery by 89%, subscription delay by 77%, end–to–end notification delay in static and dynamic routing by 47% and 58% respectively, and the lengths of output queues of brokers in dynamic routing paths by 59%.

1 INTRODUCTION
Content–based Publish/Subscribe (CPS) systems are used for many–to–many communication among loosely coupled distributed entities. A publisher publishes data in form of notifications, while a subscriber registers its interest in form of a subscription (set of filters or predicates) to receive notifications of interest [4, 10, 13, 27]. In broker–based CPS systems, a dedicated overlay network, formed by a set of inter–connected brokers, is used to connect publishers and subscribers while keeping them anonymous from each other [4, 13].

Publish/subscribe is an active area of research due to its increasing popularity and gradual adoption in different application domains [27]. It is the communication substrate in social networking systems [12, 28], business process monitoring [22], software defined networking [26], massive multi–player online games [1], and many commercial applications [9, 12, 16, 17, 19]. A subscription is broadcast in the overlay network and saved in routing table of each broker in order to form a subscription–tree. Upon receiving a notification, a broker calculates next destination–paths by matching contents in the notification with filters in saved subscriptions. This technique is called the content–based or filter–based routing using Reverse Path Forwarding [4, 8]. A CPS system in this paper refers to a subscription–based publish/subscribe system that uses a dedicated broker–based overlay network for content–based routing [5].

Most CPS systems use acyclic overlay topologies, which provide single routing path and offers limited flexibility to deal with network conditions like load imbalance, and link congestion. To stabilize a CPS system, subscribers are shifted from overloaded to less loaded brokers in a network area. Unfortunately, finding less loaded brokers requires extra in-broker processing and generates additional network traffic, which exerts more load on the system. This not only keeps a system unstable for quite sometime until the load shift process is complete but also causes loss of messages [6]. Cyclic overlay networks are expected to improve performance and throughput by offering multiple paths among publishers and subscribers. When a link congestion or load imbalance is detected, the best available path can be selected to offer dynamic routing. Although multiple paths may be available, at most one routing path, activated by a subscription–tree, can be used to route notifications [5]. If a link is congested, no alternative routing paths are available for content–based routing. The available multiple links can be exploited to generate new content-based routing paths and avoid shifting subscribers, however, this requires an intelligent algorithm to search for alternative routing paths and then make a large number of updates in routing tables, which is costly and not scalable for large network settings (cf. Sec. 2). This indicates that for dynamic routing, acyclic and cyclic overlays suffer from almost the same limitations. Ideally, for high throughput and scalability, dynamic routing should be achieved without requiring updates in routing tables. Notification routing in cyclic overlay networks has received a little attention and, to the best of our knowledge, no CPS system offers dynamic routing. In addition to the limitation of one subscription–tree, the traditional CPS systems have more fundamental issues. For example, each subscription should be uniquely identified to avoid loops. Notifications should carry identifications of matching subscriptions to identify routing paths. Extra inter-broker messages, and larger lengths of routing paths with no support for dynamic routing (cf. Sec. 2).

This paper introduces the first CPS system, OctopiS, which offers inter–cluster dynamic routing. The system is based on a purpose–built topology called Structured Cyclic Overlay Topology (SCOT) generated from Cartesian product of graphs (cf. Sec. 3). We use a novel clustering technique to divide SCOT into groups of brokers (i.e., clusters). Classifications of clusters, brokers, and links are introduced to define structuredness of a SCOT (cf. Sec. 4). OctopiS exploits the structuredness to generate subscription–trees of shortest lengths and do not require unique identifications to detect loops. The system offers inter–cluster dynamic routing without making...
updates in routing tables (cf. Secs. 4 & 5). The proposed Static Notification Routing (SNR) algorithm uses subscription–trees to send notifications to interested subscribers, while Dynamic Notification Routing (DNR) algorithm reduces delivery delay by offering inter–cluster dynamic routing when congestion is detected. In summary, the contributions of this paper are as follows. (i) Sec. 2 identifies issues surrounding the CPS systems that use cyclic overlays. (ii) Sec. 3 introduces how Cartesian product of graphs can be used to formally describe SCOT. Additional constraints are introduced to optimize SCOT for content–based routing. (iii) Sec. 4 describes a clustering approach with additional classifications for topology elements to prevent loops in SCOT. This section also introduces a lightweight bit–vector mechanism to identify target clusters for inter–cluster dynamic routing. (iv) A subscription broadcast algorithm that generates subscription–trees of shortest lengths in a clustered SCOT is described in Sec. 5. (v) Details on static routing (by SNR algorithm) and dynamic routing (by DNR algorithm) are provided in Sec. 6. (vi) Comparison with state–of–the–art identification–based routing is discussed in Sec. 7. We describe related work in Sec. 8, and conclude in Sec. 9.

2 BACKGROUND ISSUES

In this section, we use Fig. 1 to discuss different issues related to content–based routing in cyclic overlays. In particular, we discuss the issue of: (i) adding a Unique Identification to each subscription to avoid cycles in subscription broadcast, (ii) Extra Inter–broker Messages (IMs) to detect and discard duplicate subscriptions, (iii) larger Lengths of Subscription–trees, and (iv) Path Identification for notification routing. In this paper, an overlay link is represented as \( l(source, destination) \), where the source and destination are message sending and receiving brokers respectively.

II Unique Identification: Content–based routing generates loops in cyclic overlays. Loops route a message indefinitely if it is not detected and discarded. The Subscription Broadcast Process (SBP) broadcasts subscriptions to form subscription–trees for notifications routing. Since multiple paths can be available, a broker may receive duplicate subscriptions (or duplicates). To solve this issue, each broker adds its unique identifier, called Broker Identification (or BID), in a subscription of the local subscriber. The subscription and BID form a network–wide unique identification, which is saved in routing tables to detect and discard duplicates.

12 Extra Inter–broker Messages (IMs): Extra IMs have to be generated to detect and discard duplicates. In Fig. 1(a), broker 4 receives the subscription of S2 from broker 5. The link \( l(5, 4) \) is added in the subscription–tree of S2, assuming that broker 4 discards a second copy of the subscription received from broker 1. Similarly, broker 5 discards a second copy of the subscription of S1 received from broker 2, assuming that the first copy was already received from broker 4 (there is, therefore, no subscription–tree link from broker 2 to broker 5). This indicates that, despite using BIDs, extra IMs are generated to detect and discard duplicates.

13 Subscription–tree Length: In an acyclic overlay, SBP generates a unique subscription–tree even if a subscription is issued multiple times (after calling unsubscribe from the same broker). However, this is not the case in cyclic overlays, where multiple subscribe calls issued from the same broker may generate multiple subscription–trees with different lengths. SBP in cyclic overlays is an uncontrolled process and selects the first available link (or broker) as the next destination. This may generate subscription–trees of larger lengths when load on the links and brokers is uneven. For example, subscription–trees of S1 and S2 in Fig. 1(a) have shortest lengths (number of hops), however, the subscription–tree of S2 in Fig. 1(b) has larger length. Presumably, the links \( l(2, 5) \) and \( l(3, 6) \) had heavy network traffic when S2 issued the subscription. Broker 6 received the subscription of S2 from broker 5 and discarded the duplicate received from broker 3. Although P and S2 are hosted by the brokers in the same region (i.e., \( R_3 \)), S2 receives notifications from P after they are processed by brokers in \( R_1 \) and \( R_2 \). Subscription–trees of longer lengths increase in–broker computation, generate extra IMs, waste network bandwidth, and cause high latency in notification delivery. Ideally, a subscription–tree should always have the shortest length, even if some links have high loads when the subscription is issued (e.g., the subscription–trees in Fig. 1(c)). To the best of our knowledge, no CPS system generates subscription–trees of shortest lengths.

14 Path Identification: The issue of loops is also relevant for delivering notifications. For example, the matching process executed at broker 6 in Fig. 1(a) indicates that a notification \( n \) from P should be forwarded onto the links \( l(6, 3) \) and \( l(6, 5) \). Broker 6 creates two copies of \( n, n_1 \) and \( n_2, \) to forward to brokers 5 and 3, respectively. S1 receives \( n_1 \) from broker 4 and S2 receives \( n_2 \) from broker 3. Unfortunately, the matching process at broker 3 indicates that the subscription of S1 matches \( n_2 \), and a copy of \( n_2, \) say \( n'_2, \) should
be forwarded to broker 2. $n'_2$ ultimately reaches S1 after being forwarded by broker 4. Again, the matching process at broker 4 identifies that S2 should also receive a copy of $n'_2$, and this process continues indefinitely until identified and stopped. Similarly, $n_1$, forwarded by broker 6 onto $l(6,5)$, is received more than once. To prevent receiving duplicates, host broker of a publisher adds BIDs, assigned to matching subscriptions, to a notification. Reverse path forwarding technique is used to route the notification along the paths identified by the BIDs and each interested broker removes its BID from the notification before forwarding it to the next brokers. Routing is stopped when no BID is left in the notification. This approach is further explained by using Fig. 1(d). For any notification $m$ from P, broker 6 creates two copies, $m_1$ and $m_2$, where $m_1$ with the BID of broker 3 is forwarded onto link $l(6,3)$, and $m_2$ with the BIDs of brokers 4 and 5 is forwarded onto link $l(6,5)$. Because of carrying an additional BID, payload of $m_2$ is greater than payload of $m_1$.

After receiving $m_2$, broker 5 removes its own BID and forwards one copy of $m_2$ to S3, and another copy to broker 4. After receiving $m_2$, broker 4 removes its own BID and forwards a copy of $m_2$ to S1. Since no BID is left, broker 4 does not forward $m_2$ any further. In a large overlay network, a notification may be received by a large number of subscribers hosted by many brokers and many BIDs may have to be added in a notification [16]. In scenarios where scalability is a major requirement, BID-based routing is a bottleneck.

### 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:

1. **Acyclic Property**: Emphasizes that the $G_{af}$ must be an acyclic graph, and
2. **Connectivity Property**: Requires that $G_{cf}$ must be a complete graph. These properties augment a SCOT with essential characteristics that are used for generating subscription-trees of shortest lengths. $V_{af}$ 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 $G_{af} \boxplus \{h\}$ is called a $G_{af}$ fiber with index $h$.

Similarly, for a singleton graph of vertex set $\{m\} \subset G_{af}$, the graph $G_{cf} \boxplus \{m\}$ is called a $G_{cf}$ fiber with index $m$. We describe the importance of using indexes in SCOT fibers in Section 4. The definitions of the fibers indicate that, for each vertex of $G_{cf}$, CPUG generates one replica of $G_{af}$, and for each vertex of $G_{af}$, CPUG generates one replica of $G_{cf}$. The number of fibers of $G_{af}$ and $G_{cf}$ is equal to $|V_{af}|$ and $|V_{cf}|$, respectively.

![Figure 2: Operands of CPUG: Left of $\square$ is $G_{af}$ which is an H-graph; right of $\square$ is $G_{cf}$ which is a triangle.](image-url)
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 does not generate extra links or nodes, since CPUG is commutative. These two properties are used for clustering and routing purposes (cf. Secs. 4–6). In Fig. 2, the left operand of $\Box$ operator, an acyclic $H$–graph, is the $G_{af}$, while the second operand $G_{cf}$ is a triangle, which is a complete graph. More details on SCOT are available in the technical report [25].

---

4 CLUSTERING FOR STRUCTUREDNESS

Parallel paths or links provided by a SCOT are not enough to handle issues presented in Sec. 2. This section describes a set of classifications for brokers and links to build a structuredness in SCOT. The structuredness divides a SCOT into uniquely identifiable groups of brokers called clusters. Another pattern of grouping divides a SCOT into multiple regions. Types for clusters, brokers, and links are used to generate subscription–trees of shortest lengths, avoid use of unique identifications to detect loops, and support inter-cluster dynamic routing (cf. Secs. 5 & 6). More details are provided in the following.

4.1 Cluster and Region

Each $G_{af}^i$ — 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 $G_{af}^i$ — fiber) when a CPUG is calculated. Similarly, each $G_{cf}^j$ — 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 $V_{af}$ that generates the region (or $G_{cf}^j$ — fiber) when a CPUG is calculated. There are $|V_{cf}|$ and $|V_{af}|$ number of clusters and regions in a SCOT, respectively. The 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 Overlay Links and Messaging

A SCOT has two types of links: (i) an intra-cluster overlay link (aCOL), and (ii) an inter-cluster overlay link (iCOL). aCOLs connect brokers in the same cluster, while iCOLs connect brokers in the same region. Messaging along aCOLs and iCOLs is referred to as intra- and inter-cluster messaging, respectively. The set of all aCOLs in a cluster $C_i$ is $\{l(x, i, z) | x, z \in V_{af} \land y \in E_{af}\}$, while the set of all aCOLs in a SCOT is $\{l(x, z, y) | x, y \in V_{af} \land x, y \in E_{af} \land y \in E_{af} \land z \in V_{cf}\}$. Similarly, the set of all iCOLs in a region $R_j$ is $\{l(x, y) | x, y \in V_{cf} \land x, y' \in E_{cf}\}$, while the set of all iCOLs in a SCOT is $\{l(x, y) | x, y' \in V_{cf} \land x, y' \in E_{cf} \land z' \in V_{cf}\}$. There are $|E_{af}|.|V_{cf}|$ number of aCOLs, and $|V_{af}|.|E_{cf}|$ iCOLs in a SCOT. A target link refers to an overlay link that is part of a notification routing path.

4.3 Classification of Clusters and Brokers

Classification or types of SCOT brokers and clusters is used for cluster-level routing of subscriptions and notifications (cf. Secs. 5 & 6). The cluster that contains the host broker of a client is the Primary or Host Cluster, while all other are the Secondary Clusters of the client. The Primary neighbours of a broker belong to the same cluster, while the Secondary neighbours are those in the same region. Primary and secondary neighbours are also called direct neighbours. This arrangement of secondary brokers requires only one iCOL to forward messages from one cluster to any other cluster. The host (or secondary) cluster of a publisher is its Target Cluster (or Target Secondary Cluster (TSC)) if the cluster hosts at least one interested subscriber. 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 is represented by $B_{x, y}$, where $x \in V_{af}$ and $y \in V_{cf}$ (from Eq. 1), and is aware of its own type (i.e., edge or inner), the types of its primary and secondary neighbours, and the types of its links (i.e., aCOLs and iCOLs).

| Bit Indexes | Bit Values |
|------------|------------|
| 7 6 5 4 3 2 1 0 | 0 0 0 0 0 1 0 0 |
| 7 6 5 4 3 2 1 0 | 0 0 0 0 0 1 0 1 |

(a) $CBV_i$ for a subscription. (b) $CBV_p$ for a notification.

---

4.4 Cluster Bit Vector

Cluster Bit Vector (CBV) is a row (vector) of bits used to identify the host cluster of a subscriber and the TSCs of a publisher. It has two contexts: (i) the subscription context $CBV_i$ identifies host cluster of the subscriber, and (ii) the publication context $CBV_p$ identifies TSCs in inter-cluster dynamic routing (cf. Sec. 6). Bits in CBV are indexed from right to left, with the index of the right most bit being zero. Each bit of CBV is reserved for a SCOT cluster where index of the bit is same as index of the cluster it represents. Since each subscriber has at most one host cluster, there is only one meaningful bit in each $CBV_i$. The $CBV_i$ in Fig. 4(a) indicates that $C_2$ is the host cluster of the subscriber (index of the bit and cluster is 2). Each
We use Fig. 2 and Fig. 3 to explain structure of a clustered SCOT.

**Subscription Broadcast Process (SBP)**

CBV of a notification in Fig. 4(b) indicates that \( C_0 \) and \( C_2 \) are the TSCs of the publisher and should receive the notification. The number of significant bits in a CBV is equal to the number of clusters in a SCOT. The SCOT in Fig. 3 has three clusters and requires only three bits in CBV to identify all possible TSCs. OctopiS saves CBV in routing tables while CBV is carried with a notification in inter-cluster dynamic routing.

### 4.5 Example

We use Fig. 2 and Fig. 3 to explain structure of a clustered SCOT. Fig. 2 shows that \( |G_{af}| = 6 \) (i.e., \( |G_{af}| = 5 \)), while \( |G_{c,f}| = 3 \) and \( |G_{c,f}| = 3 \). There are 3 clusters and 6 regions where each cluster is identified by \( C_f \), where \( i \in \{0,1,2\} \). Each region is identified by \( R_j \), where \( j \in \{a,b,c,d,e,f\} \). The set of all aCOLs in \( C_0 \) (or \( C_0^f - \text{fiber} \)) is given as:

\[
\{l[(a,0),(b,0)](a,b) \in V_{af} \cap \forall a \in E_{af}\}.
\]

Similarly, the set of iCOLs in \( R_a \) (or \( G_{c,f}^f - \text{fiber} \)), is given by:

\[
\{l[(a,x),(a,y)](x,y) \in V_{cf} \cap \forall x \in E_{cf}\}.
\]

All brokers in regions \( R_a, R_e, R_d, \) and \( R_f \) are edge, while in \( R_b, R_c \) and \( R_e \) are inner brokers. \( B(a,0) \) and \( B(c,0) \) are the primary, while \( B(b,1) \) and \( B(b,2) \) are the secondary neighbours of \( B(b,0) \). \( \langle |E_{af}||V_{cf}| + |V_{df}||E_{cf}| \rangle \) is 33.

### 5 SUBSCRIPTION BROADCAST

The **Subscription Broadcast Process (SBP)** in traditional CPS systems is a one-step process in which a subscription reaches every broker of a cyclic overlay. However, OctopiS performs SBP in two-steps. We use this approach to exploit structuredness of SCOT for a controlled SBP to generate subscription–trees of shortest–lengths, avoid using unique identification for each subscription, and prevent loops and extra IMs (eliminating I1, I2, and I3). Each broker of a subscriber’s host cluster performs the two-steps, while the host broker of the subscriber sets its cluster index bit in CBV to 1 before forwarding a subscription. CBV is for S1, S2, and S3 in Fig. 5 are CBV[0], CBV[1], and CBV[1], respectively.

Each subscription in clustered SCOT has two states: (i) primary state, and (ii) secondary state. In the first step, a subscription is forwarded to brokers in a subscriber’s host cluster and state of the subscription is primary. No loops occur, as the host cluster is a replica of an acyclic factor \( G_{af} \) (recall the acyclic property of SCOT graph operand) and the subscription broadcast is similar to in an acyclic overlay, which generates subscription–tree of the minimum length. No unique identification is needed, and no duplicates appear. aCOLs of the host cluster are added in the subscription–tree, which has the maximum length \( \leq \text{diam}G_{af} \). In the second step, each broker in the host cluster of the subscriber changes state of the subscription to secondary and forwards it to secondary neighbours. The secondary neighbours do not forward a secondary subscription to any other broker. In this step, all iCOLs are added to the subscription–tree and the maximum length is \( \leq \text{diam}G_{af} + 1 \). A subscription with its CBV is saved as \( \langle \text{subscription, last hop, CBV} \rangle \) tuple in routing tables. A broker can find state of the subscription by examining CBV saved with the subscription. For a primary subscription, the index of the bit with value 1 should be same as the cluster (or broker) index. As shown in Fig. 5, in the first step of SBP for S1, \( B(a,0) \) forwards the subscription to the primary broker \( B(b,0) \), and in the second step, to the secondary brokers \( B(b,1) \) and \( B(a,2) \). Each broker of \( C_0 \) repeats the first and second steps to generate the subscription–tree of S1. Similarly, the subscription–trees of S2 and S3 are generated. Contrary to traditional CPS systems, SBP in OctopiS is controlled, which always generates a unique subscription–tree for a subscription issued from a broker. Each broker is aware of its type (i.e, primary or secondary), and each cluster is treated as an exclusive acyclic overlay. SBP uses this information to forward a subscription onto specific links to brokers (e.g., in Fig. 5). Uneven load in brokers or links does not effect structure or length of a subscription–tree. This pattern of subscription broadcast does not generate duplicates.

**Algorithm 1** provides more details about the two-steps of SBP. The state attribute of a subscription has two values: **PRIMARY** and **SECONDARY**. The host broker of a subscriber sets the host cluster index bit \( CBV_f \) to 1 (lines 4-7), and forwards the subscription to direct neighbours. The \( \text{isPrimary}(n) \) method checks the type of the next broker and the state attribute of the subscription is set accordingly (lines 10-11). The subscription message is saved on the primary and secondary brokers (line 15).

**Figure 5:** The two-step subscription forwarding process in the SCOT in Fig. 3. \( C_0 \) is the host cluster of S1 and S2, while \( C_1 \) is the host cluster of S3. The solid arrows indicate part of the subscription–trees generated in the first step and the dashed arrows indicate part of the subscription–trees generated in the second step.

The subscription–trees in clustered SCOT require no updates in secondary brokers when a subscriber relocates to some other broker in the same cluster as the second step of SBP does not change. This provides robust fault tolerance when a broker fails for some reasons (fault tolerance in OctopiS is not within scope of this paper).

### 6 NOTIFICATION ROUTING

This section describes our static and inter–cluster dynamic routing approaches in clustered SCOT. OctopiS uses cluster–based static routing to deliver notifications using subscription–trees of shortest lengths, and switches to dynamic routing when congestion is detected. We also outline state–of–the–art BID-based routing in unclustered SCOT.

#### 6.1 BID–based Static Routing

An unclustered SCOT has no types or groups of brokers and links, which is prerequisite to use BID–based routing algorithm. To identify routing paths in an unclustered SCOT, BID–based routing algorithm requires a notification to carry BIDs assigned to matching...
Algorithm 1: scotSBPs

Input: s : a subscription message;
Output: DL : a list of next destinations that should receive s ;

1 /* PRIMARY state indicates host cluster of the subscriber */
2 if s.state = PRIMARY ∨ isHostBrokers then
3 /* Host broker sets the index bit for the subscription. i is the cluster index. */
4 if isHostBrokers then
5 s.CBV_i ← setBitOni;
6 s.hostBroker ← false;
7 s.state ← PRIMARY;
8 /* forward subscription to all primary neighbours */
9 foreach n ∈ NeighbourList – Sender do
10 if isPrimary[n] ≠ true then
11 s.state ← SECONDARY;
12 s.next ← n;
13 DL.addn;
14 /* subscription is in SECONDARY state, not forwarded to any broker */
15 RT.inserts ;

submissions. SBP in unclustered SCOT is uncontrolled and lengths of routing paths satisfy the relation:

\[
\max (d(x_1, y_1, x_2, y_2)) \leq (|G_{cf}| \cdot \text{diam}G_{af} + 1 - 1)
\]

where \(x_1, y_1\) and \(x_2, y_2\) are any brokers. Increase in payload due to adding BIDs is an important concern because normally a BID is formed by a combination of IP address and a broker level unique identifier [10]. For a large network, where a large number of brokers may host interested subscribers, BID–based routing is not scalable because of additional payload and impeding in-broker processing in matching process [4, 20].

As Algorithm 2 shows, the host broker of a publisher adds BIDs of the matching subscriptions in bidList attribute of a notification \(n\) (lines 3–7). splitBIDs method creates hash map, which uses a next destination path as hash key and the list of BIDs of brokers down to the next destination path as object of the hash key. The presence of an object localBID in the hash map indicates that each local subscriber with a matching subscriptions should receive a copy of \(n\) (lines 11–16). Afterwards, a copy of \(n\) with the corresponding list of the remaining BIDs are forwarded onto the next destinations (lines 18–21).

6.2 Cluster–based Static Routing

Static Notification Routing (SNR) algorithm uses subscription–trees of shortest lengths for routing notifications in a clustered SCOT. The algorithm deals with two scenarios. (i) The host cluster of the publisher is the only target cluster, and therefore 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. (ii) At least one interested subscriber is hosted by a secondary cluster and therefore lengths of the routing paths satisfy the relation:

\[
\max (d(x_1, y_1, x_2, y_2)) \leq (\text{diam}G_{af} + 1)
\]

where \(x_1, y_1\) and \(x_2, y_2\) are any brokers in a clustered SCOT. The host broker of a publisher forwards a notification onto target aC0Ls and iC0Ls (recall that a target link is part of a routing path to next destination). The notification is routed to interested subscribers in the host and TSCs of the publisher using reverse path forwarding technique. No loops appear and no path identifications are required because each cluster is an acyclic overlay (eliminating 14). In Fig. 6, a notification from \(P1\) propagates onto the routing path in \(C2\) to reach \(S4\). \(B(f,2)\) creates two additional copies of the notification to forward to \(B(f,1)\) and \(B(f,0)\) in the TSCs. For \(S1\) and \(S2\) only one copy of the notification is forwarded to \(B(f,0)\) to avoid duplicates. Similarly, notifications from \(P2\) and \(P3\) are forwarded. Note that \(P2\) has no primary target cluster, while \(P3\) has no TSC.

In Algorithm 3, getDistinctSubs uses matching subscriptions to return a list of distinct next destinations (line 5). The method getDistinctSubs assures that only one copy of notification \(n\) is forwarded onto a next destination link. The else part (lines 7–9) handles intra–cluster messaging when \(n\) is received by a broker other than the host broker of the publisher. nextDistinct aCOLs provides

Algorithm 2: pubBidn

Input: n : a notification message;
Output: DL : a list of next destinations that should receive n ;

1 /* host broker of publisher adds BIDs */
2 if isHostBroker then
3 IS ← getInterestedSubs;
4 foreach s ∈ IS do
5 n.bidList ← s.bid;
6 n.hostBroker ← false;
7 /* next destination based split list of BIDs */
8 nextBIDs ← splitBIDs.n.bidList;
9 /* send notifications to local interested subscribers */
10 if localBID ∈ nextBIDs then
11 localSubs ← getLocalSubs;
12 foreach client ∈ localSubs do
13 n.next ← client;
14 DL.addn;
15 nextBIDs ← nextBIDs – localBID;
16 /* send notifications to next brokers in routing paths */
17 foreach nextHop ∈ nextBIDs.keys do
18 n.next ← nextHop;
19 n.bidList ← nextBIDs.getnextHop;
20 DL.addn;
a list of the next distinct destinations to forward $n$ in the same cluster.

Figure 6: Notification routing in SCOT shown in Fig. 3. $C_2$ is the host cluster of the publisher $P_1$, while $C_1$ is the host cluster of the publishers $P_2$ and $P_3$. The set of interested subscribers for $P_1$, $P_2$, and $P_3$ are $\{S_1, S_2, S_3, S_4\}$, $\{S_1, S_2, S_4\}$ and $\{S_3\}$, respectively. The set of target clusters of $P_1$, $P_2$ and $P_3$ are $\{C_0, C_1, C_2\}$, $\{C_0, C_2\}$ and $\{C_1\}$, respectively. Red, black, and brown (dashed, solid) arrows indicate intra- and inter-cluster notification routing for $P_1$, $P_2$, and $P_3$, respectively.

Algorithm 3: scotSNRn

```
Input: $n$: a notification message;
1 return DL: a list of next destinations that should receive $n$;
2 /* forward $n$ onto all target links */;
3 if isHostBroker then
4   /* get distinct subscriptions to identify next target links */;
5   destinations ← getDistinctSubsn;
6   n.hostBroker ← false;
7 else
8   /* forward $n$ onto distinct target aCOLs */;
9   destinations ← nextDistinct_aCOLsn;
10 foreach $d$ ∈ destinations do
11     n.next ← $d$;
12     DL.addn
```

6.3 Inter–cluster Dynamic Routing

Dynamic routing refers to the capability of a network system to alter a routing path in response to overloaded or failed links and/or routers. Multiple techniques have been proposed to offer 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 [21]. IP addresses may be used to form clusters in a network area where the same network mask requires a single routing entry. However, these techniques are not applicable in broker-based CPS systems because brokers in these systems are aware of only their direct neighbours and destination 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 when one or more target inter–cluster overlay links (i.e., iCOLs) are overloaded and notifications start queuing up in the output queues. Overloading and subsequent queuing can happen in two cases: (i) when a broker is not able to process high volume of outgoing notifications and become overwhelmed, and (ii) when bandwidth is limited. As SBP generates only one subscription–tree per subscription, dynamic routing is difficult in broker-based CPS systems and never supported before. We introduce a unique approach, which deviates from traditional reverse path forwarding technique and uses structuredness of the proposed topology (i.e., SCOT) along with subscription–trees of the matching subscriptions to offer inter-cluster dynamic routing. Our approach is scalable because it does not require updates in routing tables and can reduce delivery delays when a large number of notifications start accumulating in the output queues.

SNR algorithm adds exclusive copies of a notification in the output queues of the target links. If a publisher generates $\gamma$ number of notifications in $t_w$ time window, and there are $\alpha$ number of target aCOLs and $\beta$ target iCOLs, the host broker of the publisher enqueue $\alpha + \beta + \gamma$ number of copies of notifications. A High Rate Publisher (HRP) with a high value of $\gamma$ can overwhelm brokers in a routing paths when SNR algorithm is used. Our inter-cluster Dynamic Notification Routing (DNR) algorithm alleviates overwhelmed brokers and selects an unoverloaded iCOL to forward a notification to a TSC. The algorithm dynamically selects an unoverloaded iCOL, which may not be part of a subscription–tree to the TSC. As parallel iCOLs are available in a clustered SCOT, one can be selected without making updates in routing tables (partially eliminating I5). DNR adds at most one copy of a notification in the congested output queues of a target link at a broker. The algorithm sets the cluster index bits in $CBV_p$ to 1 to identify those TSCs that are not forwarded the notification due to overloaded target iCOLs. $CBV_p$ is added in the header of the notification, called the $CBV_p − Notification$ (or $CBV_p − N$). $CBV_p − N$ is an indication for a broker that the routing is dynamic, and a copy of the notification should be forwarded to TSCs if unoverloaded target iCOLs are available. Using this technique, DNR algorithm keeps the length of the congested output queues of an overwhelmed broker to a minimum, and the load of forwarding the notification is shifted to other brokers using the heuristic that unoverloaded target iCOLs are available down the routing path. octopiS uses Eq. 3 to find whether an output queue is congested, and DNR algorithm should be activated.

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

$Q_{in}$ and $Q_{out}$ are the number of notifications that enter into or leave the output queue in the time window $t_w$, respectively. The term $Q_{in} + Q_{out}$ 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 $Q_t$ is 0. octopiS 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. Inter-cluster dynamic routing by DNR algorithm is further explained in the following with help of three cases.

Case 1 – Unoverloaded Target iCOL: When the output queue of at least one target iCOL is uncongested (i.e., the corresponding link is unoverloaded), DNR algorithm does not enqueue a notification in the congested output queues of the target iCOLs. Instead, only one copy of the notification is enqueued in the uncongested output queue of...
the unoverloaded target iCOL. If more than one unoverloaded target iCOL is available at a broker, $CBV_p - N$ is forwarded onto the target iCOL, which has the least value of $Q_t$. The number of notifications added to the output queues of the target links in $t_n$ interval is $\alpha + \beta$, where $\alpha$ is the number of unoverloaded target iCOLs and $\beta < \beta$. Fig. 7(b) indicates that the iCOL $l(b, 2, b, 0)$, which forwards notifications from $B(b, 2)$ to $C_0$, is overloaded. After finding the overloaded iCOL from the LST, $B(b, 2)$ sets the index bit of $C_0$ to 1 and adds $CBV_p - N$ (with $CBV_p, 001$) in the output queue of the unoverloaded target iCOL $l(b, 2, b, 1)$ to forward $CBV_p - N$ to $B(b, 1)$. Since the output queue of the iCOL $l(b, 1, b, 0)$ is uncongested, $B(b, 1)$ sets the index bit of $C_0$ to 0, removes the $CBV_p$ from the notification as all index bits in $CBV_p$ are zero, and forwards the notification to $B(b, 0)$ and $B(c, 1)$. The notification is forwarded to $S2$, $S3$, and $S4$ using their subscription–trees. However, to avoid overloaded link $l(b, 2, b, 0)$, DNR algorithm deviates from the subscription–tree of $S1$ and dynamically selects the iCOL $l(b, 1, b, 0)$ for routing without making updates in routing tables. To forward one notification from P to interested subscribers, each of SNR and DNR algorithms generate 6 IMs, although DNR algorithm excluded one overloaded link, the dynamic routing path for $S1$ contains 4 brokers, which is one more than the number of brokers in path adopted by SNR algorithm.

**Case II – All Target iCOLs Overloaded:** When the output queues of all target iCOLs are congested and at least one output queue of a target aCOL is uncongested, DNR algorithm uses the target aCOLs to find unoverloaded target iCOLs. $CBV_p - N$ is added in the congested output queue of the target aCOL. The load of forwarding the notification to TSCs is shifted to the next primary broker. The number of notifications added to the output queues of the target links in $t_n$ interval is $\alpha \gamma$ (here $\beta$ is zero as no copy of the notification is added into the congested output queues of the target iCOLs). Fig. 7(c) indicates that the two target iCOLs, $l(b, 2, b, 1)$ and $l(b, 2, b, 0)$, are overloaded, and notifications are sent only to $B(a, 2)$ and $B(a, 0)$ in the host cluster of P. Since two unoverloaded target aCOLs are available, the target aCOL with the least $Q_t$ is selected to forward $CBV_p - N$ with $CBV_p, 011$ (presumably, aCOL $l(b, 2, c, 2)$ has least value of $Q_t$). As the target iCOLs are not overloaded at $B(c, 2)$, the notification is routed to the TSCs after removing $CBV_p$. To forward one notification in this case, DNR algorithm generates 4 IMs, while SNR algorithm generates 6 IMs. Furthermore, two overloaded iCOLs are excluded from the dynamically generated routing paths.

**Case III – All Target Links Overloaded:** If all target links are overloaded, $CBV_p - N$ is forwarded onto the least loaded target iCOL. Because of the possibility of having more overloaded aCOLs down the routing path, $CBV_p - N$ is not added into the output queue of a target aCOL even if it is the least overloaded link. The number of notifications added to the output queues of the target links in $t_n$ interval is $\alpha + 1 \gamma$. Fig. 7(d) shows that $CBV_p - N$ is forwarded onto (presumably) the least overloaded link $l(b, 2, b, 1)$ with $CBV_p, 001$. As the target iCOL $l(b, 1, b, 0)$ is also overloaded, $B(b, 1)$ forwards $CBV_p - N$ onto the target aCOL $l(b, 1, c, 0)$. The overloaded link $l(c, 1, c, 0)$ is the only target iCOL available to forward the notification to TSC $C_0$, DNR algorithm is unable to find unoverloaded target iCOL and the notification has to be forwarded onto the overloaded iCOL $l(c, 1, c, 0)$. The number of IMs generated by SNR and DNR algorithms in this case are 6 and 5 respectively. Although DNR algorithm successfully avoided the overloaded iCOL $l(b, 2, b, 0)$, the notification has to be forwarded onto another overloaded target iCOL $l(c, 1, c, 0)$. Additionally, the generated dynamic routing path for $S1$ has 3 additional brokers. Algorithm 4 provides further details about the inter-cluster dynamic notification routing in 0ct0p15. Upon receiving a notification n, the overwhelmed broker prepares a list of local subscribers interested in n (line 5). A copy of n for each subscriber is added in the next destinations list (lines 7-9). Next, n, a list of overloaded target iCOLs is prepared to set index bits in $CBV_p$ (lines 13-18). Note that the index bit of the current notification routing cluster is not set (line 17) because this information does not effect inter-cluster dynamic routing. A copy of n for each unoverloaded target link is added in the next destination list DL (lines 22-24). In the end, the algorithm shows how three cases are handled. The first if–statement handles the Case II when all target iCOLs are overloaded and an unoverloaded target aCOL is available (lines 26-29). The second condition (in the else-if block) is valid when an unoverloaded target iCOL is available to carry $CBV_p$ (lines 30-33). The else block handles the case when all target links are overloaded and a copy of n is added in DL to sent onto least overloaded target iCOL $\eta 2$.

DNR is a best–effort algorithm and depends on the subscription–trees laid–on by interested subscribers in a target cluster. The algorithm does not guarantee finding an unoverloaded target iCOL, even if one exists. For example, in Fig. 7(d), DNR algorithm does not use the unoverloaded link $l(b, 2, a, 2)$ for inter-cluster dynamic routing.
because the link is not a target aCOL. Forwarding the CBV\(\mu\) − \(N\) on such links can generate loops among different clusters [25]. DNR algorithm currently does not support intra–cluster dynamic routing, and this is the reason that OctopiS eliminates IS only partially. \textit{intra–cluster dynamic routing} is part of the future work.

7 EVALUATION

We implemented SNR, and DNR algorithms in OctopiS, developed on top of a publish/subscribe tool PADRES [10]. For a comparison with state–of–the–art, we also implemented BID–based routing in PADRES. Importantly, we created a subscription–based publish/subscribe version of PADRES, as the tool supports advertisement–based semantics [20]. SNR and DNR algorithms exploit structuredness of clustered SCOT, while BID–based routing algorithm uses unclustered SCOT.

7.1 Setup

Fig. 8 shows factors of the SCOT topology \(S_e\), which we used for evaluation and comparison of SNR, DNR, and BID–based routing algorithms. \(G_{af}\) factor of \(S_e\) is an acyclic topology of 15 brokers (5 inner brokers (vi, vii, viii, ix and x) and 10 edge brokers), while \(G_{cf}\) factor has 5 brokers, which generates \(|V_{cf}|−1\) number of secondary neighbours for each broker in \(G_{af}\). This results in 25 inner brokers and 50 edge brokers (for a total of 75 brokers), forming 5 clusters and 15 regions in \(S_e\). OctopiS was deployed on a cluster of 35 physical computing nodes, where each node had one quad core processor of 2.4 GHz with 4 GB RAM, and running 64-bit JDK on Linux OS. Each broker was loaded in a separate instance of JVM with 1 GB initial memory. One dedicated high throughput Gigabit Ethernet switch was used for connectivity. \(S_e\) was deployed in such a way that the primary and secondary neighbours of each broker were always on different computing nodes.

![Figure 8](image_url)

**Figure 8:** Left operand of \(\Box\) operator is the \(G_{af}\) factor while the right operand is the \(G_{cf}\) factor of \(S_e\). The \(G_{cf}\) generates five replicas of \(G_{af}\).

Stock datasets are commonly used to generate workloads for evaluations of CPS systems [11]. We used a dataset of 500 stock symbols from Yahoo Finance!, where each stock notification had 10 distinct attributes. This high dimension data require high computation for filtering information during in–broker processing. Subscriptions were generated synthetically. We randomly distributed publishers and subscribers, where each subscriber registered one subscription with 2% selectivity.

7.2 Metrics

Through experiments with real world data, we evaluated OctopiS using primitive metrics, such as subscription, notification, and matching delays.

\textbf{Subscription delay}: The subscription (forwarding) delay is the maximum time elapsed as a subscription reaches brokers in an overlay network. A subscription is expected to take less time to reach brokers in a close proximity to the host broker of a subscriber than to brokers in a far–off region. Since the SBP in a clustered SCOT generates subscription–trees of shortest lengths, it is important to
measure the difference between the average subscription delays in unclustered and clustered SCOT.

**Notification delay:** The notification delay measures end-to-end latency from the time a notification is generated to the time it is received by a subscriber. As the average length of subscription–trees for BID-based routing is higher than a clustered SCOT for SNR and DNR algorithms, knowing the difference in latencies in these two cases is a worthwhile area of inquiry.

**Matching delay:** The matching delay is the time taken to find subscriptions that match with the notification contents. BID-based routing is expected to have less matching delay as the matching is done at only the brokers which host publishers and interested subscribers.

**Inter-broker Messages (IMs):** The number of IMs depends on the lengths of subscription–trees, as well as the relative distance between publishers and subscribers. As the average length of subscription–trees in clustered SCOT is less than in unclustered SCOT, the number of IMs generated by SNR and DNR algorithms is expected to be less than BID-based routing algorithm.

Aggregation techniques like covering are developed for acyclic overlays to reduce size of routing tables. These techniques can be used with clustered SCOT as each cluster is an acyclic overlay. However, for a comparison with state-of-the-art BID-based routing, covering is not considered in evaluation because no covering technique for cyclic overlays is reported in literature.

### 7.3 Results

The results presented in this section cover three important aspects of the evaluation: (i) **Subscriber Scalability,** which studies the behaviour of the routing algorithms when the number of subscribers increases, while the number of publishers remains constant; (ii) **Publisher Scalability** is about study of the algorithms with a varying number of publishers and constant number of subscribers; (iii) **Burst Scenario,** in which an HRP starts sending notifications at a high rate and causes congestion in the output queues.

**Subscriber Scalability:** We gradually increased the number of subscribers from 500 to 10000, and used 100 publishers, each sending 1000 notifications at the rate of 60 notifications per minute (npm). All publishers start sending the notifications in the first 5 seconds after all the subscriptions register. We adopted this pattern of generating controlled workload to count IMs and notifications received by the subscribers. Fig. 9(a) shows that the number of IMs generated by SBP in clustered $S_c$ (legends SNR, and DNR) are 89% less than in unclustered $S_c$ (legend BID). There are two reasons for this significant difference: (i) larger average lengths of subscription–trees, and (ii) extra IMs generated to discard duplicate subscriptions in unclustered $S_c$. The average length of the subscription–trees generated by OctopiS is 14% less than the PADRES. Furthermore, almost 80% of the generated IMs are used to detect and discard duplicates in unclustered $S_c$ [20]. OctopiS uses a pattern of subscription broadcast that does not generate duplicates and extra IMs. Fig. 9(b) shows that the average subscription delay in clustered $S_c$ is 77% less than the unclustered $S_c$. The difference is due to the larger lengths of subscription–trees and extra IMs generated in unclustered $S_c$ to detect duplicate subscriptions. Importantly, there is no difference in subscription delay when SNR and DNR algorithms are used, since SBP does not use inter-cluster dynamic routing even if some output queues are congested. This approach generates subscription–trees of shortest lengths even if some links are overloaded. The average subscription delay in unclustered and clustered $S_c$ is nearly constant. Fig. 9(c) shows that the end-to-end notification delay in SNR and DNR is less than BID–based routing approach. More specifically, SNR and DNR algorithms reduce the notification delay by 47% when compared with BID–based routing algorithm. There are three reasons for this difference: (i) the larger length of subscription–trees in BID–based routing, (ii) high payload due to carrying BIDs of the matching subscriptions, and (iii) extra processing by brokers to prepare and split lists of BIDs down the routing paths to find the next destinations. In large networks with thousands of brokers in multiple data centres (e.g., in [14]), a notification may have to carry thousands of BIDs to identify routing paths. OctopiS does not require carrying BIDs with notifications (a lightweight bit vector $CBV_p$ is added when the notification routing is dynamic). Delay in static and dynamic routing is almost the same as this experiment does not generate dynamic routing paths for small workloads. Fig. 9(d) shows that the number of IMs generated by the same number of notifications (1000 per publisher) in SNR and DNR algorithms is 12% less than BID–based approach. Again, the difference in the number of IMs is due to larger lengths of subscription–trees generated by the BID-based subscription routing. The difference between the number of generated IMs decreases with the increase in the number of subscribers because of the possibility of having less forwarder–only brokers in a routing path. Fig. 10(a) shows that the average matching delay in BID–based routing is 25% less than SNR and DNR algorithms, and increase almost linearly. In BID-based notification routing, only the host brokers of publishers and interested subscribers execute the matching process, while no matching occurs at the intermediate brokers [20], while SNR and DNR algorithms performs matching at each broker, which results in larger matching delays. The difference between matching delays in the three algorithms decreases with increase in the number of subscribers, which decreases forwarder–only brokers.

**Publisher Scalability:** We increase the number of publishers from 100 to 500 (each sending 500 notifications at a fixed rate of 100 npm) with 3000 subscribers. Fig. 10(b) shows that the average notification delay in SNR and DNR algorithms is 42% less than BID–based routing algorithm. The difference is due to the larger length of subscription–trees and higher payload due to carrying BIDs with notifications. Because of the difference in lengths of subscription–trees, the number of IMs generated by BID–based routing is 36.4% higher than SNR and DNR algorithms (Fig. 10(c)). The average matching delay in SNR and DNR algorithms is 265% higher than BID–based algorithm (Fig. 10(d)). BID–based routing matches a notification only at the host brokers of publishers and interested subscribers, while SNR and DNR algorithms match a notification at each broker of a routing path. As the number of subscribers is constant, the matching delays introduced by three algorithms is linear.

**Stability Analysis:** The stability analysis tells how quickly a CPS system converges to a normal state after an HRP finishes sending notifications. To study this behaviour, we set the value of $\tau$ to 10 and $t_p$ to 50 milliseconds. We used 5000 subscribers, and 100 publishers each issued 2000 notifications at the rate of 60 npm. 0.2% of the subscribers subscribe to receive notifications from the HRP. We
execute three simulations with the HRP sending 100K notifications at rates of 100K, 80K, and 60K npm. All clusters of $s_e$ are target of the HRP. Furthermore, the HRP and interested subscribers are hosted by different brokers exerting more load on iCOLs and aCOLs. The burst continued for 60 to 100 seconds depending on the rate of the HRP. Each point in the graphs in Fig. 11 is a maximum delivery delay of 1000 notifications received in a sequence. This approach helps in analysing the stability without graphing too many points. Each simulation is run until all notifications are received. Fig. 11 shows that DNR algorithm stabilized @ctop15 before SNR algorithm, while BID–based routing algorithm is not able to stabilize PADRES for the same workload. On average, in the first 18 minutes and 30 seconds, the maximum delay of a notification (out of 1000) is the same in the three routing algorithms and no tendency toward stability is observed. This indicates that, due to the high rates of notifications, the state of the system (@ctop15) does not converge to normal until the condition $CE < 1$ is maintained for some time (on average, 16 minutes and 40 seconds for the three simulations). DNR starts stabilizing @ctop15 before the other two algorithms. The average value of $Q_t$ of target links at the brokers in routing paths of notifications from the HRP when DNR is used is 48% less than SNR and 59% less than BID–based routing algorithms. There are 5 target clusters of the HRP and DNR tends to add the minimum number of copies of a notification when the output queues of the target links are congested. This decreased the length of $Q_t$, when compared with SNR and BID–based algorithms. The average notification delay in DNR algorithm when the notification rate is 100K is 13%, and 58% less than SNR, and BID–based routing algorithms, respectively. Similar improvements, when the rates are 80K, 60K npm, are 12.1% and 53%, and 11% and 49.2%, respectively. On average, for the three simulations, DNR algorithm stabilizes the system 239 seconds before SNR algorithm and generates only 0.32% IMs more than SNR and 17.2% less than BID–based routing algorithms. This shows that our approach of adding at most one copy of a notification when the output queues are congested significantly reduces delivery delay and queue length. Analysis of the collected data indicate that the number of notifications that have delivery delays less than 1 second in DNR, SNR and BID–based routing algorithms are 44.2%, 39% and 28%, respectively. We also conducted several experiments with HRP in each cluster and sending notifications with different rates. The performance difference between DNR and SNR algorithms decreases with an increase in the number of HRPs that start sending notifications simultaneously. This indicates that the improvement
due to inter-cluster dynamic routing in OctopiS diminishes when more congested output queues and overloaded iCOLs appear.

8 RELATED WORK

Content-based routing in distributed broker-based CPS systems has been the focus of many research efforts. SIENA [5], JEDI [7], Rebeca [24], PADRES [10], Kyra [2], and MEDYN [3] are just few examples. Notifications routing in cyclic overlays has got a little attention from the research community. SIENA introduces a notifications routing scheme for cyclic overlays and detects duplicates using BIDs. Latency-based distance-vector algorithms generate routing paths for advertisement- and subscription-based CPS systems [4]. However, the algorithms do not generate subscription–trees of shortest lengths and, despite multi-path overlay networks, dynamic routing is not supported. Subscription–trees generated for CPS systems in [4] can be improved by periodically sending subscription messages to find links with the minimum latency; however, this refinement generates extra traffic in the network and is infeasible for a large network settings. Li et al [20] offers dynamic routing without making updates in routing tables in advertisement-based publish/subscribe systems. A large number of IMs are generated in the advertisement broadcast process to detect duplicates. Unique path identifications are added in notifications for routing to interested subscribers. Dynamic routing relies on brokers with intersecting routing paths generated by multiple advertisements matching one subscription. Therefore dynamic routing for a subscription (or subscriber) matching with one advertisement is not possible. Furthermore, intersecting routing paths are not always possible even if a subscription matches with multiple advertisements [20]. The probability of having brokers that publish intersecting routing paths also depends upon the number of multiple advertisements matching a subscription. Finally, a subscription has to be delivered multiple times to the broker that publishes more than one advertisement intersecting that subscription [25]. Shuping et al [18] propose an approach which divides an overlay into interconnected clusters to apply content-based and destination based intra- and inter-cluster routing. Algorithms for routing using shortest paths are developed, however, the approach requires embedding routing path identifications, which increases payload and consume network bandwidth inefficiently. A broker has to be aware of all other brokers in its host cluster, which increases topology maintenance cost. Dynamic routing requires updates in routing tables to generate alternative routing paths, which increases network traffic and delivery delays. OctopiS offers inter-cluster dynamic routing without requiring updates in routing tables. Thanks to the structurenessed of clustered SCOT overlays, which provide multiple inter-cluster routing paths. OctopiS neither generates redundant IMs or duplicates in SBP nor requires BIDs. Notifications are delivered using subscription–trees of shortest lengths and without requiring unique identifications to identify routing paths to prevent loops. Brokers in SCOT overlay are decoupled and aware of their direct neighbours only, which requires a very low topology maintenance cost. All these properties make OctopiS scalable and suitable for large content-based routing networks.

9 CONCLUSION

In this paper, we present the design and evaluation of the first (subscription–based) CPS system, OctopiS, that offers inter–cluster dynamic routing of notifications without requiring updates in routing tables. The system uses a novel structured cyclic topology SCOT, which is constructed applying optimizations on Cartesian product of two graphs. A homogeneous clustering technique is introduced, which divides a SCOT into similar identifiable blocks of brokers. We further exploit the structureness of clustered SCOT to generate subscription–trees of shortest lengths, without generating extra IMs to detect and discard duplicates. A static routing algorithm SNR uses subscription–trees of shortest lengths to send notifications to interested subscribers, while DNR algorithm offers inter–cluster dynamic routing using a lightweight bit vector mechanism. Both algorithms do not require a global knowledge of an overlay topology and dynamic routing is activated when congestion is detected in the output queues of the target links. The evaluation of SNR and DNR algorithms, and comparison with BID–based routing algorithm indicates that OctopiS scales better than state–of–the–art and suitable for large network settings.

REFERENCES

[1] Cesar Cañas, Kaiwen Zhang, Bettina Kemme, Jorg Kienzle, 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.
[2] Fengyun Cao and J.P. Singh. Efficient event routing in content-based publish-subscribe service networks. In INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, volume 2, pages 929–940 vol.2, March 2004.
[3] 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.
[4] Antonio Carzaniga. Architectures for an event notification service scalable to wide-area networks, phd thesis, politecnico di milano. 1998.
[5] 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.
[6] Alex King Yeung Cheung and Hans-Arno Jacobsen. Load balancing content-based publish/subscribe systems. ACM Trans. Comput. Syst., 28(4):9:1–9:55, December 2010.
[7] G. Cugola, Elisabetta Di Nito, and A. Fuggetta. The jidi event-based infrastructure and its application to the development of the opus wfm. Software Engineering, IEEE Transactions on, 27(9):827–850, 2001.
[8] Yogen K. Dalal and Robert M. Metcalfe. Reverse path forwarding of broadcast packets. Commun. ACM, 21(12):1040–1048, December 1978.
[9] Cooper et al. Pnuts: 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] Yogeshwer Sharma et al. Wormhole: Reliable pub-sub to support geo-replicated internet services. In NSDI 15, pages 351–366, Oakland, CA, May 2015. USENIX Association.
[13] Patrick Th. Egerster, Pascal A. Feller, Rachid Guerroumi, and Anne-Marie Kermarrec. The many faces of publish/subscribe. ACM Comput. Surv., 35(2):114–131, June 2003.
[14] Keith Hamilton. Web solutions platform (wsp) event system. In Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, DEBS ’12, pages 12–19, New York, NY, USA, 2012. ACM.
[15] Richard Hammack, Wilfried Imrich, and Sandi Klavzar. Handbook of Product Graphs, Second Edition. CRC Press, Inc., Boca Raton, FL, USA, 2nd edition, 2011.
[16] Microsoft Bing Pulse (http://pulse.bing.com/).
[17] Google Cloud Pub/Sub (https://cloud.google.com/pubsub/).
[18] 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.
[19] Jay Kreps, Neha Narkhed, and Jun Rao. Kafka: a distributed messaging system for log processin. NetDB Workshop, pages 01–07, 2011.
[20] 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.
[21] Deepankar Medhi and Karthikeyan Ramasamy. Network Routing: Algorithms, Protocols, and Architectures. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
[22] Vinod Muthusamy. Flexible Distributed Business Process Management. PhD thesis, University of Toronto, 2011.
[23] E. Parsonage, H.X. Nguyen, R. Bowden, S. Knight, N. Falkner, 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.
[24] Helge Parzyjegla, Daniel Graff, Arnd Schrister, Jan Richling, and Gero Muhl. From Active Data Management to Event-Based Systems and More: Papers in Honor of Alejandro Buchmann on the Occasion of His 60th Birthday, chapter Design and Implementation of the Rebeca Publish/Subscribe Middleware, pages 124–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
[25] Muhammad Shafique. Eliminating tight coupling using subscriptions subgrouping in structured overlays. CoRR, abs/1611.08743, 2016.
[26] Muhammad Adnan Tariq, Boris Koldehofe, Sukanya Bhowmik, and Kurt Rothermel. Pleroma: A sdn-based high performance publish/subscribe middleware. Middleware ’14, pages 217–228, New York, NY, USA, 2014. ACM.
[27] Sasu Tarkoma. Publish / Subscribe Systems: Design and Principles. Wiley Publishing, 1st edition, 2012.
[28] Kaiwen Zhang, M. Sadoghi, V. Muthusamy, and H.-A. Jacobsen. Distributed ranked data dissemination in social networks. In Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on, pages 369–379, July 2013.