DYNATOPS: A Dynamic Topic-based Publish/Subscribe Architecture

Ye Zhao
Dept. of Information and Computer Science
University of California, Irvine
yez@uci.edu

Kyungbaek Kim
Dept. of Electronics and Computer Engineering
Chonnam National University, South Korea
kyungbaekkim@jnu.ac.kr

Nalini Venkatasubramanian
Dept. of Information and Computer Science
University of California, Irvine
nalini@ics.uci.edu

ABSTRACT

Emerging societal scale notification applications call for a system that is able to efficiently support simple, yet changing subscriptions for a very large number of users. In this paper we propose DYNATOPS, a dynamic topic-based pub/sub architecture that provides efficient scalable societal scale event notifications for dynamic subscriptions via distributed broker networks. In DYNATOPS, users are moderately repositioned on brokers and brokers are moderately repositioned on the overlay structure for efficient event notifications, to adapt to the publications and subscription dynamics. In contrast to existing self-organized techniques, the broker network reconfiguration in DYNATOPS is executed in a planned manner utilizing a cost-driven reconfiguration process. With extensive experiments, we observe that under highly dynamic subscriptions DYNATOPS can still maintain an efficient dissemination structure that provides 30% less notification delay and overhead in general, and a reconfiguration cost reduction of 80% as compared to other state-of-the-art systems.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

Keywords

topic based, publish/subscribe, dynamic subscriptions

1. INTRODUCTION

The recent years have witnessed the growth of societal scale notification systems that have penetrated multiple aspects of our day-to-day life. Key examples include mobile social networks (e.g. Twitter and Foursquare), geomarketing (e.g. shopalerts.att.com), traffic and weather alerts (e.g. www.wunderground.com and weather.gov), emergency response (e.g. www.gdacs.org/alerts and www.mywarn.com), etc. These systems are large (e.g. Twitter is estimated to reach 500 million users in 2012 and currently handles over 340 million tweets daily), geographically distributed and largely subscription based. The notion of subscriptions in such systems is often simple, and instantiated by an average citizen – deals for local shops, traffic alerts for freeways, weather conditions for zipcodes, and location check-ins from friends. Moreover, we observe an emerging trend that the interests of such users, i.e. subscriptions, are short lived (duration of minutes or hours) and change dynamically based on users' locations or contexts.

For instance, mobile subscribers are often interested in events and information within their immediate vicinity; the dynamically changing location is a key aspect of what constitutes a subscription and consequently a relevant notification in this case. In disasters (e.g. earthquakes), people's information needs (i.e. subscriptions) change with the evolution of the disaster event (e.g. pre-disaster or post-disaster). An individual's region of interest, i.e. subscription, changes as he/she moves into or out of disaster region; people are anxious to check the safety of their beloved ones and the changing locations or statuses of families or friends cause subscription changes as well. Such shifts in notification needs calls for a system that is able to efficiently support simple, yet changing subscriptions for a very large number of users.

Publish/subscribe has for long been a popular communication paradigm to provide customized notifications to users in a distributed environment due to its loose coupling between the information providers (i.e. publishers) and consumers (i.e. subscribers). Pub/Sub systems are either content-based where subscribers receive notifications when the content of the message matches their interest [6, 35, 32, 8, 16] or topic-based [30, 40, 36, 20] when the "topic" of the publication matches their interest.

One of the main incentives for content-based pub/sub is to enable delivery of relevant and meaningful notifications through rich and expressive subscription languages. Here, sophisticated subscription management strategies are usually required to determine matching subscribers and subsequently route only the relevant events; runtime overheads for subscription insertion/removal and event matching are much higher than topic-based implementations [5, 34, 28]. Over the last decade, significant progress has been made in the design of pub/sub systems to address scalability and efficiency of content-based pub/sub [28, 13, 21, 14, 22], largely motivated by the needs of enterprise systems. The design of pub/sub systems taking into account frequently changing subscriptions is beginning to receive interest, especially in the context of content-based pub/sub systems where the emphasis is on designing novel subscription languages and schemas. For example, [28] proposed a parametric subscription language in content-based pub/sub systems to reduce broker runtime complexity in the event of a subscription update. The key idea is to insert variables into subscriptions which may be updated frequently so that only variable update is performed during subscription update. Techniques to manage the overheads due to event matching and dynamic subscription updates are however still a concern.

We conjecture that a topic-based pub/sub system can form the basis of an efficient architecture to deal with the simple, yet changing notification needs of a large number of users; for example by routing events through group multicast to peers that match subscription topics. Subscription insertion/removal and event routing on a broker can be easily done in $O(1)$ time. Indeed, topic-based pub/sub systems have been widely deployed in contexts where events divide naturally into groups (i.e. "topics") and efficient information notification is demanded [23, 9]. More recent topic-based pub sub systems, e.g. Twitter, have scaled to a very large number of users.

In this paper, we propose DYNATOPS, a novel topic-based pub/sub system that provides efficient scalable event notifications for dynamic subscriptions. To enable rapid notifications that can scale to the societal level, in the DYNATOPS system (Section 3), users connect to a distributed broker network that is built on top of a structured overlay for subscription management and publication distribution. DYNATOPS incorporates two unique broker management mechanisms to address subscription dynamicity. First, it uses a **similarity-based user placement** (Section 4) technique to map DYNATOPS users to nearby brokers. Under fast changing subscriptions such as those caused by changing locations of users, conventional user placement policies show a dramatic increase in their subscription management overhead. The proposed technique efficiently groups users sharing similar interests to be managed by the same set of brokers, to effectively alleviate the overhead with regard to subscription updates among brokers. Our experimental results (Section 6) indicate over 40% reduction in brokers' subscription management overhead as compared to other state of the art techniques under highly dynamic subscriptions.

Second, more importantly, to provide efficient event notifications, DYNATOPS incorporates a broker reconfiguration process (Section 5) that is both **subscription and structure-aware**. Unlike existing topic-based pub/sub systems where topology changes are triggered in a self-organizing manner to reflect subscription changes, we propose a cost-driven reconfiguration process that changes topology in a planned manner. The key intuition behind our approach is that reconfiguration of the broker network is achieved by merely moving broker nodes to positions in an overlay structure where routing is already highly optimized - this reduces the effort for reconfiguration in a large scale system. Furthermore, we trigger the reconfigurations only when they are likely to have high utility. The combination of these strategies allows us to reduce notification delay and overhead by 30%, and reduce the cost of reconfiguration by over 80% as compared to other state of the art techniques under highly dynamic subscriptions (Section 6).

2. RELATED WORK

Conventional pub/sub systems assume that clients join the broker environment in one of the following ways: (a) connecting to any broker with no restrictions [27, 35], or (b) connecting to the closest broker [16, 24, 29]. [7] is the first to investigate client placements that optimize delivery delay and system load. However, the work focused on the placement of a few known publishers under fixed subscribers' subscriptions and broker overlay topologies. Such schemes are inefficient in societal scale notification applications where (a) publishers are many (in general the entire user set) and (b) subscriptions change frequently based on user interests. In this paper, we explore a similarity-based user placement that takes into account both the dynamics of user subscriptions and broker load. The proposed technique can effectively reduce subscription management overhead and improve notification efficiency under dynamic subscriptions of users.

Building and reconfiguring overlay networks that take into account nodes' subscriptions for dissemination efficiency has been explored in both content-based and topic-based pub/sub systems. In content-based paradigm, Sub-2-Sub [39] is a content-based protocol that clusters nodes according to their subscriptions to construct a ring for each attribute. [37] proposed a self-organizing algorithm to cluster brokers that supposedly will be target for the same events in the near future. [43] discussed primitives that reconfiguration protocols need to implement to ensure high availability with minimum disruption under topology changes. In topic-based paradigm, many recent topic-based pub/sub systems [38, 42, 20, 36, 31, 19] build and maintain their overlays based on brokers' subscriptions. For rendezvous-based pub/sub systems that maintain topic-routing trees, Magnet [38] clusters nodes with similar subscriptions on a skewed DHT, and explores a customized routing to reduce the number of relay nodes in the multicast trees. When a node's subscription changes, the node needs to rejoin the DHT to be placed onto a new position based on its new subscription. On the other hand, [42, 20, 36, 31, 19] build relay-free overlays without topic rendezvous and explore gossip-based dissemination and/or in-cluster flooding to disseminate event notifications. Tera and StAN [36, 31] creates dedicated topic overlays for each topic. A node joins overlays for the topics that it subscribes to by connecting to a node already in the overlay. SpiderCast and TCO [20, 19] build a single unstructured overlay that strives to maximize clustering of nodes according to their interest in topics. The focus of system is to keep low node degrees while maintaining the topic-connected property as the foundation of the relay-free routing. This is achieved by the neighbor maintenance routine in SpiderCast and the overlay construction algorithm in TCO, both of which can trigger an overlay topology reconfiguration when a node's subscription changes. PolderCast [42] maintains a ring structure for each topic and combines deterministic dissemination over a ring with probabilistic dissemination similar to gossiping. Its overlay management mechanism also triggers topology updates whenever node churns or subscription changes.

While all the above systems trigger topology changes to reflect subscription changes in a self-organizing manner, in this paper we design DYNATOPS with a different philosophy. We argue that in societal scale notification applications where subscriptions are short lived and frequent change is the norm, the topology reconfiguration should be incorporated in a more systematic and planned manner in the sys-

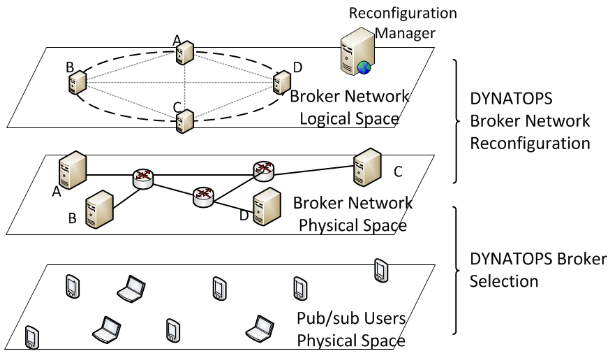
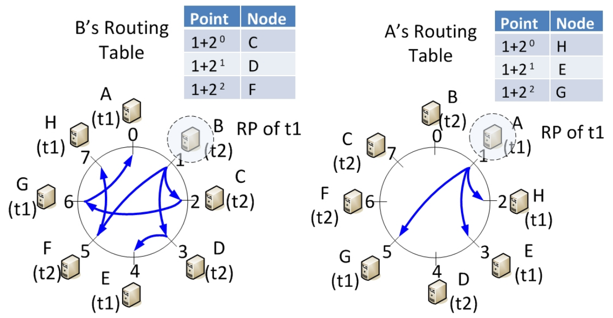

Reconfiguration Manager
A
B
D
C
Broker Network Logical Space
DYNATOPS Broker Network Reconfiguration
A
B
D
C
Broker Network Physical Space
DYNATOPS Broker Selection
Pub/sub Users Physical Space

Figure 1: DYNATOPS system overview



(a) overlay before reconfiguration (b) overlay after reconfiguration

Figure 2: example of broker network reconfiguration

tem design. This is to reduce the reconfiguration overhead from frequent subscription changes that we already know will happen. This is achieved through built-in mechanisms that detect when sufficient changes have occurred and that deal with those changes through planned overlay reconfigurations. We design a cost-driven reconfiguration process that detects when sufficient changes have occurred and deals with those changes through planned overlay reconfigurations that are likely to have high utility.

3. DYNATOPS OVERVIEW

Figure 1 shows the DYNATOPS system where pub/sub users connect to brokers for topic subscriptions and event publications. To provide scalable and efficient pub/sub services, DYNATOPS maintains a broker network based on structured overlays (e.g. Chord DHT [25]); Brokers are configured onto the overlay as overlay nodes with unique nodeIDs. Similar to [30, 40], DYNATOPS constructs a Rendezvous-based broker network and maintains independent topic routing trees to route event messages of different topics: each topic has a Rendezvous Point (RP), which is an overlay node responsible for the hashed key of the topic name in the DHT, and the RP serves as the root of the corresponding topic routing tree. Every broker determines which trees to join based on the subscriptions of users that connects to it. A broker must join the topic tree when at least one of its users subscribes to the topic, and leaves the tree when all of them unsubscribe to minimize notification overhead. For convenience, we use "broker subscribes to the topic" to refer to the first case and "broker unsubscribes to the topic" to refer to the second case. A topic routing tree spans all brokers that subscribe to the topic and it is created in a top-down fashion such that the routing path from the root to each of the brokers in the tree is consistent with the default routing path by the underlying key-based routing algorithm. An example of topic routing tree using Chord is shown in Fig. 2a. There are 8 brokers uniformly placed into the structure with their mapped IDs shown on the ring. We considered 2 topics $t1$ and $t2$, and they are hashed to IDs 1 and 4 respectively. Each broker is interested in only one topic. The arrows in the ring indicate the topic routing tree for $t1$ based on the routing table of each broker (we only show one of them in the figure). For event publications, the published messages are first forwarded from publishers to their topic RPs. Then the RP nodes disseminate the messages along the topic routing trees to every single broker that subscribes to the topic.

3.1 Dynamic Mappings

The centerpiece of the DYNATOPS system are two efficient dynamic mapping algorithms in response to users' and brokers' dynamic subscriptions: 1) a simple distributed *similarity-based user placement* algorithm that maps pub/sub users to pub/sub brokers; 2) the *DYNATOPS broker network reconfiguration* algorithm running on a Reconfiguration Manager to manage the broker network structure (i.e. mapping of brokers to overlay nodes in logical space).

The reconfiguration manager is a logically centralized entity that monitors the pub/sub environment, e.g. rates of event publications and subscription changes at brokers, and intelligently adapts the broker network structure for fast and efficient event notifications. It only communicates with brokers periodically with minimal message overhead. To alleviate the concern that the reconfiguration manager may pose a bottleneck or a single point of failure, it can be distributed over multiple servers using a distributed overlay similar to that proposed in our earlier implementations [1].

The aim of the user placement algorithm is to reduce brokers' subscription changes so as to reduce subscription management overhead and alleviate the demand for broker network reconfiguration (described later) when the pub/sub users' subscriptions are dynamic. This is done by dynamically aggregating users with similar subscriptions to be managed together by the same set of brokers.

However, as brokers' subscription change topic routing trees in rendezvous-based pub/sub may involve unrelated relay brokers that are not themselves interested in the topic but that reside on the routing path from the root to subscribed brokers (see example in Figure 2a). The unrelated relay brokers cause excessive delay and overhead in event notifications. Our aim is to eliminate/reduce the unrelated relay brokers. In DYNATOPS we explore the possibility of dynamically altering the broker network structure to match the underlying (possibly changing) subscription needs - i.e a *Structure and Subscription aware Broker Reconfiguration (SSBR)* to map brokers to overlay nodeIDs. The goal of the reconfiguration is to intelligently adapt the broker network based on updated pub/sub environment to construct efficient topic routing trees. Figure 2b shows an example illustrating how DYNATOPS reconfigures the broker network for efficient event notifications based on updated broker subscriptions. We can see that before reconfiguration, event notification in $t1$ takes 3 unrelated relays (Broker C, D and F are not interested in the topic 1), and 3 maximum hops

(Broker A receives the published event after 3 hops). After reconfiguration, the tree changes along with the change of the mapping and of the routing table on each broker. Note, the topic trees are still constructed following the basic routing rules of Chord. For $t1$, the unrelated relays becomes 0 and the maximum hop becomes 1 after reconfiguration.

Figure 3 shows the system architecture for DYNATOPS broker network reconfiguration. DYNATOPS brokers handle the publications and subscriptions from its pub/sub users through the pub/sub manager component. They also periodically report their pub/sub states (e.g. number of publications and subscription summaries) to the reconfiguration manager. Based on the received information, the reconfiguration manager performs a cost-driven reconfiguration computation to seek a balance between potential performance improvement and reconfiguration cost from the network reconfiguration. The reconfiguration manager then coordinates the overlay structure configuration with the structure manager component on each broker through its configuration controller component.

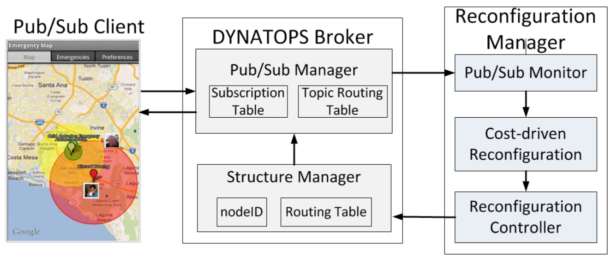


Figure 3: DYNATOPS Broker System Architecture

4. USER PLACEMENT STRATEGY

The core of our user placement strategy is a grouping algorithm that takes as an input the location and subscription of a user, and outputs the broker that is responsible for hosting the user's subscriptions. The goal of the algorithm is to ensure that users with similar subscriptions are grouped and managed by the same set of brokers. The grouping of similar users brings two benefits to the pub/sub system. First, it minimizes the subscription management overhead at brokers. Subscription management overhead incurs in topic-based broker networks when brokers' subscriptions change. The intuition here is that by clustering similar users each broker subscribes to topics that interests all/most of the users it hosts, so that the subscription states of the brokers are more stable and less dependent on a single user's subscription changes. Furthermore, when a user subscribes to a new topic, it is more likely that the topic has also been subscribed by users with similar interests, so that the topic has been subscribed by the broker hosting the user.

Second, it reduces the potential number of brokers that require to subscribe to each single topic. In rendezvous-based pub/sub implementations, the number of brokers subscribing to a topic directly reflects to the size of the topic routing tree and subsequently impacts on the event notification delay and overhead on the specific topic. This is particularly true for DYNATOPS because the broker overlay is configured to eliminate/reduce unrelated relay brokers in topic routing trees so that the tree size is more dependent on the number of brokers subscribed to the topic.

Algorithm 1: User Join Protocol

(1) the user sends the join request to a random broker he knows during bootstrap with its subscriptions. In the case that the user has a connected broker already but changes its subscriptions, the join request is piggybacked to the subscription change message.
(2) the requested broker calculates the *user-broker utility* $Util_{u,b}$ for all its visible brokers (explained later).
(3) the requested broker replies to the user with the broker ID (address) that yields the largest utility value in the last step.
(4) the user joins the system by connecting to the selected broker.

Let P denote the set of topics, we evaluate the similarity of random user u to broker b in their subscriptions $s_u, S_b \subseteq P$ by a *user-broker similarity* metric, $Sim_{b,v}$, which is defined as the normalized size of their intersection, in the range $[0, 1]$. Formally, $Sim_{b,v} = |S_b \cap s_v|/|s_v|$. Our scheme is flexible and can accommodate other similarity metrics (e.g. Jaccard similarity), however, their analysis is out of the scope of the paper. Similarity based user placement will select for each user u a hosting broker that yields the highest user-broker similarity among all brokers. However, user placement based only on similarity metrics may result in severe load imbalance at brokers, especially when users' subscriptions are highly skewed (e.g. when most users only subscribe to a few very popular topics). That is, a few brokers have to host the majority of users due to high similarity of their subscriptions. Hence, to alleviate load imbalance, we also take into account a *load* metric, L_b, to reflect the load level of a broker, with range $[0, 1]$. In this work we assume homogeneity of brokers and define broker load as the normalized amount of local user subscriptions managed by a broker. Formally, $L_b = \sum_{u \in U_b} |s_u| / \sum_{u \in U} |s_u|$. A *user-broker utility* combining the similarity and load metrics is defined as $Util_{u,b} = Sim_{b,u} - w_l \cdot L_b$, where w_l is the relative weight for the load metrics.

The crux of user placement is the user join protocol which is executed every time a user subscribes to a new topic, or unsubscribe an existing topic. We present the join protocol in Algorithm 1.

Ideally the grouping algorithm requires each broker to acquire the global information on the subscriptions and loads of other brokers. This will incur significant overhead when the network size is large. This message overhead is considered as part of the subscription management overhead in DYNATOPS. Hence, we explored two techniques to avoid excessive overhead: 1) instead of maintaining the real-time state of subscriptions and loads of other brokers, each DYNATOPS broker only updates the information every T_{update} period; 2) we explored a clustering technique to divide brokers into clusters and only let brokers in the same cluster exchange the subscription and load information. When a user joins, the broker running the algorithm only evaluates utilities for brokers from the same cluster. We show in our experimental evaluation that a small cluster size (e.g. 10 brokers in a cluster) is good enough for the grouping algorithm to outperform existing techniques.

5. COST-DRIVEN BROKER NETWORK RECONFIGURATION

DYNATOPS aims to reconfigure the broker network to minimize event notification overhead and delays. We provide a formal modeling of the notification performance metrics and use it to drive the broker network reconfiguration process.

Let B denote the set of brokers, and $G(V, E)$ denote the structured overlay topology that DYNATOPS builds its broker network atop. A vertex $v \in V$ is an overlay node with a fixed nodeID. This corresponds to a fixed node position in the overlay geometry too. Its edges to other vertices are links to other overlay nodes. We assume $|V| = |B|$. DYNATOPS configures the broker network by specifying the mapping of the set of brokers B onto the set of overlay nodes V (i.e. nodeIDs) in $G(V, E)$. Therefore, we can define the configuration as a mapping matrix, $X = \{x_{b,v}\}$, between B and V, such that:

$$x_{b,v} = \begin{cases} 1 & \text{if broker } b \text{ is mapped to overlay node } v; \\ 0 & \text{otherwise.} \end{cases}$$

Let S_b be the subscriptions of broker b. In the rendezvous-based pub/sub implementation, given the overlay configuration X and $S = \{S_b : \forall b \in B\}$ as the set of broker subscriptions, the topic routing tree for each topic can be determined easily. The tree is rooted at the topic rendezvous and spans all overlay nodes whose mapped broker subscribes to the topic.

To formalize the performance of the pub/sub system, we formulate the ***event overhead***, o_p, for an event in topic p as the total number of forwarded messages required to disseminate it from the RP to all brokers that subscribe to topic p. This overhead can be divided into two parts: (a) ***subscriber overhead***, o_p^{sub}, as the number of messages consumed by brokers that subscribe to the topic; and (b) ***relay overhead***, o_p^{relay}, as the number of the messages consumed by intermediate unrelated relays who *have not* subscribed to the topic. Meanwhile, we formulate the ***broker event delay***, $d_{b,p}$, for an event in topic p to reach a specific subscribed broker b as the delay to forward it from the topic RP to that broker, and it can be easily approximated by the RTTs between brokers during each hop along the path. Given the set of brokers that subscribe to topic p, we formulate the ***event delay*** as the cumulative broker event delays for all brokers in the set, as: $d_p = \sum_{b:p \in S_b} d_{b,p}$. Note that in the above formulation we opt out the overhead and delay for forwarding the event from its publisher to the RP. This is because we assume any user in the system can be a potential event publisher (regardless of his subscriptions) so that the average path length from publisher to RP is independent of broker network configurations.

To evaluate the performance of event notifications of the pub/sub by taking into account events in all available topics during a time span $(t_0, t_0 + T)$, we define ***cumulative relay overhead*** and ***cumulative delay*** under a broker network configuration X as follows:

$$O^{relay}_{(t_0,t_0+T)}(X) = \sum_{p \in P} \sum_{\tau=t_0}^{t_0+T} \delta_p(\tau) \cdot o_p^{relay}(\tau) \quad (1)$$

$$D_{(t_0,t_0+T)}(X) = \sum_{p \in P} \sum_{\tau=t_0}^{t_0+T} \delta_p(\tau) \cdot d_p(\tau) \quad (2)$$

where $\delta_p(t)$ is the number of events of topic p in time slot t, and $o_p^{relay}(t)$ and $d_p(t)$ are event relay overhead and event delay of the topic routing tree in time slot t. Note that if the brokers' subscriptions are static during the evaluation period, then all the topic routing trees are static, subsequently o_p^{relay} and d_p are static and the above equations can be rewritten as:

$$O^{relay}_{(t_0,t_0+T)}(X) = \sum_{p \in P} N_p^{(t_0,t_0+T)} \cdot o_p^{relay} \quad (3)$$

$$D_{(t_0,t_0+T)}(X) = \sum_{p \in P} N_p^{(t_0,t_0+T)} \cdot d_p \quad (4)$$

where $N_p^{(t_0,t_0+T)} = \sum_{\tau=t_0}^{t_0+T} \delta_p(\tau)$ is the total number of events of topic p during the period.

Now we define our pub/sub performance metric during an evaluation period $(t_0, t_0 + T)$ under configuration X as the ***Notification Cost***, $C_{(t_0,t_0+T)}(X)$, as follows:

$$C_{(t_0,t_0+T)}(X) = O^{relay}_{(t_0,t_0+T)}(X) + w_d \cdot D_{(t_0,t_0+T)}(X) \quad (5)$$

where $w_d \geq 0$ is a relative weight of delay performance to overhead performance.

5.1 Reconfiguration Process

A question needs to be answered is *when and how the broker network should be reconfigured.* Since reconfiguration is not cost-free – it may incur both management overhead and potential disruption of event notifications, it is not plausible to reconfigure the broker network whenever subscriptions change on brokers. To avoid frequent reconfigurations, we define *reconfiguration free period*, T_{free}, as the minimum period between two consecutive reconfigurations in DYNATOPS. Intuitively, if the benefit of making a reconfiguration cannot justify the cost of it, the broker network should not be reconfigured. Every T_{free} period, DYNATOPS performs a cost-driven reconfiguration process to evaluate the benefit of reconfiguration against its cost.

Let X be the broker network configuration before reconfiguration, and X' be the new configuration. The benefit of reconfiguration is evaluated as the gain in $C_{(t_0,t_0+T_{free})}$, which reflects the efficiency of event notifications. Formally,

$$Bft(X, X') = C_{(t_0,t_0+T_{free})}(X) - C_{(t_0,t_0+T_{free})}(X') \quad (6)$$

The cost of reconfiguration is the overhead associated with the transition of the broker network from its old configuration X to its new configuration X'. We designed a reconfiguration protocol for efficient broker network transitions in DYNATOPS. We formulate the reconfiguration cost as the upper bound we derived for the message overhead of the protocol to reconfigure the broker network over a Chord DHT (See Section 5.1.2 for the protocol and the discussion on its overhead). Assume $\Gamma(X, X')$ is the number of brokers that have a different mapping in configuration X' compared to X. The reconfiguration cost is:

$$Cst(X, X') = \begin{cases} 0 & \text{if } X' = X; \\ O(|V| + log^2(|V|) \cdot \Gamma(X, X') & \\ +|P|log(|V|) \cdot \Gamma(X, X')) & \text{otherwise.} \end{cases} \quad (7)$$

```
Algorithm 2: reconfiguration process
    Step1-online monitoring:
    online monitoring the event publications and subscription
    changes on brokers to estimate $C_{(t_0,t_0+T_{free})}(X)$.
    Step2-reconfigurability test:
    fast judging the demand of broker reconfiguration from
    $C_{(t_0,t_0+T_{free})}(X)$. If demand is low, return to step1;
    Otherwise, continue to step 3.
    Step3-reconfiguration computation:
    calculating $X'$ to maximize $Bft(X,X') - Cst(X,X')$. If
    $Bft(X,X') \leq Cst(X,X')$ return to step1; Otherwise,
    continue to step4.
    Step4-reconfiguration protocol:
    coordinating the brokers to transit to $X'$.
```

Algorithm 2 shows the DYNATOPS reconfiguration process that is periodically executed by the reconfiguration manager.

The reconfiguration manager monitors and estimates the rate of event publications as well as subscriptions to calculate $C_{(t_0,t_0+T_{free})}$. We apply an exponential filter to estimate publication rate in topic p for the next period from the observation in current period:

$$\overline{R}^{k+1}_{pub}(p) = (1-\alpha_{pub})\overline{R}^{k}_{pub}(p) + \alpha_{pub}R^{k}_{pub}(p) \tag{8}$$

where $\overline{R}^{k+1}_{pub}(p)$ is the estimated publication rate in topic p for the $(k+1)^{th}$ period, and R^{k}_{pub} is a monitored publication rate for the $(k)^{th}$ period. α_{pub} is the parameter for the filter. In DYNATOPS we set $\alpha_{pub} = 0.3$.

Since it is hard to estimate the dynamics of brokers' subscriptions $S(t)$ for each time slot t in the next evaluation period $(t_0, t_0 + T_{free})$, we simply consider a static S equals $S(t_0)$ as an approximation. Better strategies could be considered by acquiring more context-awareness of the pub/sub users.

The reconfigurability test aims to prune out unnecessary reconfiguration computations when estimated reconfiguration demand is low for the next reconfiguration free period. The intuition here is that when the broker network is already fairly optimized or the event notification cost is low, the marginal benefit of reconfiguration won't be able to justify the cost of it. Hence, we compare the estimated $C_{(t_0,t_0+T_{free})}(X)$ with a cost threshold to decide the necessity for reconfiguration computation. A reconfiguration computation is desired only if it is larger than the threshold. In this paper, we set the threshold to $\beta \cdot maxCst(X,X')$, where $maxCst(X,X')$ is the maximum possible reconfiguration cost in the broker network by equation 7 and $\beta > 0$ is a control parameter (we set it to 0.1 in our implementation).

5.1.1 Reconfiguration Computation

DYNATOPS performs reconfiguration computation to determine optimal topology configuration X' that maximizes $Bft(X,X') - Cst(X,X')$. We prove in [4] that the problem is NP-hard to solve. Instead of solving the problem directly, we first present a sub-problem and propose an efficient algorithm for the problem. We define the basic reconfiguration problem without taking into account the reconfiguration cost: ***Structure and Subscription aware Reconfiguration (SSBR)*** as follows.

$SSBR(G(V,E), S, N_p^{(t_0,t_0+T)})$: Being aware of the overlay structure $G(V,E)$, brokers subscriptions S and $N_p^{(t_0,t_0+T)}$ as the number of event publications in each topic during a period, solve the following optimization problem:

$$arg\min_X C_{(t_0,t_0+T)}(X) \tag{9}$$

where $C_{(t_0,t_0+T)}(X)$ is the notification cost defined in equation 5.

The above problem is still NP hard unless we consider the special case where only delay is considered (i.e. $w_d = \infty$) and delay between two brokers is strictly proportional to the number of hops in the overlay (In this case, the problem can be efficiently solved and an optimal solution is available). We present a greedy algorithm with upper bound complexity $O(|P|^2|V|^2log(|V|))$ (see [4] for proof) for the general case. The algorithm iteratively improves the solution given an initial broker network configuration. We call the algorithm SSBR-Greedy and show it in Algorithm 3. In each iteration, the algorithm explores the neighborhood $Nb(X)$ (we will define later) of the current best configuration X and finds the best neighboring configuration that minimizes the cost function in equation 5. Let $\widetilde{X'}$ be the best among $X' \in Nb(X)$. If $\widetilde{X'}$ is superior to X, then the best configuration moves from X to $\widetilde{X'}$. Otherwise, a local optimal configuration is reached and the algorithm stops. Each iteration the algorithm finds a better configuration with a better $C_{(t_0,t_0+T_{free})}$ than the one of last iteration. They form an improvement path in $C_{(t_0,t_0+T_{free})}$. The algorithm returns all the configurations found on the improvement path.

The neighborhood, $Nb(X)$, of a configuration X is a set of configurations that directly derivable from X. One immediate available neighborhood of X can be derived by swapping the mapping in X of any two brokers. However, this neighborhood contains $|V|(|V|-1)/2$ configurations, leading to great computational overhead for the algorithm to examine in each iteration. To improve the algorithm efficiency for online computation, we examine a different yet smaller neighborhood with only $|V|-1$ configurations.

We introduce the concept of *Broker SSBR Cost*. The broker SSBR cost, c_b evaluates each broker for their contributions to the total SSBR cost in equation 5. It consists of two parts: broker overhead cost, $c_b^O(X)$, and broker delay cost, $c_b^D(X)$. They are defined as follows:

$$c_b^O(X) = \sum_{p\in P} N_p^{(t_0,t_0+T)} \cdot z_{b,p}(X) \tag{10}$$

where $z_{b,p}(X)$ is a 0-1 variable which equals to 1 if b is an unrelated relay in the routing tree of topic p, and 0 otherwise.

$$c_b^D(X) = \sum_{p\in S_b} N_p^{(t_0,t_0+T)} \cdot d_{b,p}(X) \tag{11}$$

It is not difficult to derive the following relationship between broker SSBR cost and the total SSBR cost of the system:

$$\begin{aligned} C_{(t_0,t_0+T)}(X) &= \sum_{b\in B} c_b(X) \\ &= \sum_{b\in B}(c_b^O(X) + w_d \cdot c_b^D(X)) \end{aligned} \tag{12}$$

In each iteration our greedy algorithm finds the broker with the highest broker cost under configuration X and construct a neighborhood $Nb(X)$ consisting of configurations that are derived by swapping the mapping of the broker

Algorithm 3: SSBR-Greedy

```
Input: $G(V,E), S, N_p^{(t_0,t_0+T)}$
Output: $List of X$ that along the improvement path of
        $C_{(t_0,t_0+T)}(X)$

$\widetilde{X}$ = initial configuration; $\widetilde{minC} = C_{(t_0,t_0+T)}(\widetilde{X})$;
$List = []$ while $\widetilde{minC} < minC$ do
    $minC = \widetilde{minC}, X = \widetilde{X}$, List.add(X);
    foreach $X' \in Nb(X)$ do
        $tmpC = C_{(t_0,t_0+T)}(X')$;
        if $tmpC < \widetilde{minC}$ then
            $\widetilde{minC} = tmpC, \widetilde{X} = X'$;
        end
    end
end
```

with the highest external overhead grade with that of another broker. Apparently the neighborhood contains only $|V|-1$ configurations. By searching new configurations that have a different mapping for the bottleneck broker who has the worst broker cost, we have a better chance to improve the total cost.

We adapt the SSBR-Greedy algorithm to solve the original reconfiguration computation problem by taking into account reconfiguration cost $Cst(X, X')$, and call the new algorithm DYNATOPS-Greedy algorithm (see Algorithm 4). It consists of two steps: At the first step, the algorithm applies the SSBR-Greedy algorithm to iteratively find new configurations that improves the $C_{(t_0,t_0+T_{free})}$: each iteration the algorithm finds a better configuration with a better $C_{(t_0,t_0+T_{free})}$ than the one of last iteration. They form an improvement path in $C_{(t_0,t_0+T_{free})}$. At the second step, the reconfiguration cost $Cst(X, X')$ of all the configurations X' along the improvement path are taken into account and the best one is selected as the output of the algorithm. The algorithm has the same computational complexity as the SSBR-Greedy algorithm.

Algorithm 4: DYNATOPS-Greedy

```
Input: $G(V,E), S, N_p^{(t_0,t_0+T_{free})}, X$
Output: $X'$ that maximize $Bft(X,X') - Cst(X,X')$

step one:
$List = SSBR - Greedy()$;
step two:
$C = C_{(t_0,t_0+T_{free})}(X)$;
$bestUtil = 0$;
$X' = X$;
foreach $\widetilde{X}$ in $List$ do
    $\widetilde{C} = A.get(\widetilde{X})$;
    $Util = C - \widetilde{C} - Cst(X,\widetilde{X})$;
    if $Util > bestUtil$ then
        $bestUtil = Util$;
        $X' = \widetilde{X}$;
    end
end
```

5.1.2 *Reconfiguration Protocol*

The control plane of a DYNATOPS broker consists of two levels: 1) the overlay level, where its overlay nodeID, neighbor table and routing table are maintained; and 2)

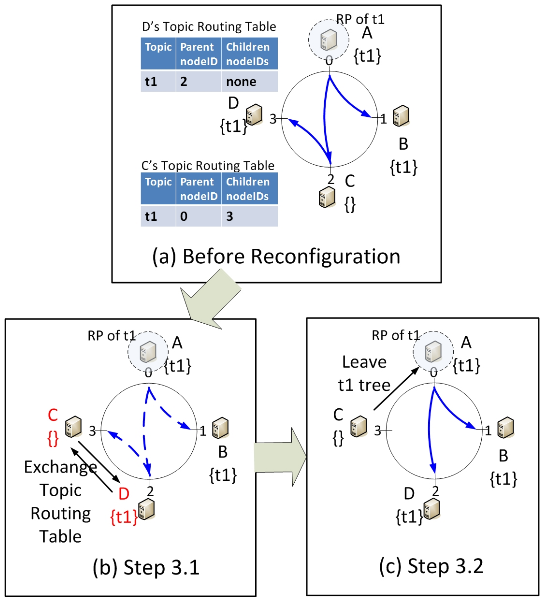


Figure 4: An example of the step 3 of the reconfiguration protocol. Solid lines in (a) shows topic t1's tree before reconfiguration. Broker "C" and "D" are reconfigured. Dashed lines in (b) shows the reconstructed tree after "C" and "D" exchange their data in step3.1. Solid lines in (c) shows the repaired topic routing tree after step 3.2.

the pub/sub level where its topic routing trees (i.e. logical parent nodes and children nodes in trees) are maintained. Each reconfiguration process has a unique version number, a successful reconfiguration will update the control plane of all brokers to the new version where both levels of the control plane must converge. On one hand, DYNATOPS relies on the maintenance protocol of the underlying structure overlay to ensure the convergence of the overlay level of the new control planes of brokers. On the other hand, we designed an efficient reconfiguration protocol to ensure the convergence of the pub/sub level of the new control planes. Taking Chord DHT as an example underlying structure, we show that the entire reconfiguration process incurs $O(|V| + (log^2(|V|) + |P|log(|V|)) \cdot \Gamma(X, X'))$ message overhead where $\Gamma(X, X')$ is the number of nodes to reconfigure. A reconfiguration of the broker network from version number $s1$ to $s2$ consists of following steps:

STEP1: The reconfiguration manager sends a RCFG_INIT message to all brokers. The message contains the new version number $s2$ of the control plane after reconfiguration. Upon receiving the message the brokers replicate their $s1$ control planes to $s2$ and maintains both instances keeping $s1$ as their default control plane to be used by data plane.

STEP2: The reconfiguration manager sends to brokers with new nodeIDs in Chord a RCFG_OVERLAY message. The brokers that receive the message leave and rejoin the Chord ring with their new nodeIDs in $s2$. Once successfully reconfigured, the brokers send a RCFG_OVERLAY_DONE message back to the reconfiguration manager. (According to [25], the step takes $O(log^2(|V|) \cdot \Gamma(X, X'))$ messages to repair the Chord ring)

STEP3: The reconfiguration manager sends to all brokers a RCFG_PUBSUB message to reconfigure the pub/sub level of the control plane. This process is divided into two substeps. An example of this process is shown in Figure 4.

- *3.1:* Each reconfigured broker (under a new nodeID in $s2$) forwards its Topic Routing Table data in $s1$ to the broker who takes its $s1$ nodeID in $s2$. This is easily done by key-based routing in the overlay without knowing the real identity of the other broker. The data corresponds to a local view of the topic routing trees it joined (see Figure 4). The brokers replace their $s2$ Topic Routing Table with the received data to restore the topic routing trees in $s2$. We illustrate in the example that the restored topic routing tree is the same as the original tree in structure. However, the tree is not necessarily consistent with the subscriptions at brokers after reconfiguration: broker "C" does not subscribe to t1 but it is a leaf node in the tree. (the step takes $O(log(|V|) \cdot \Gamma(X, X'))$ messages)

- *3.2:* To repair the inconsistency, each reconfigured broker compares its Topic Routing Table with its subscriptions, and initiates tree leave/join requests to fix its incorrect participation/absence in topic trees. The request messages are forwarded to the root of the tree and a top-down fashion update of the tree is incurred. Upon completion of this step, all the topic routing trees in $s2$ correctly reflect both the routing and subscriptions states of the broker network. (the step takes $O(|P|log(|V|) \cdot \Gamma(X, X'))$ messages)

STEP4: The reconfiguration manager sends to all brokers a RCFG_FINISH message. Once the message is received, the brokers make $s2$ as their default control plane and nullify $s1$ when all brokers switched to $s2$. (the step takes $O(|V|)$ messages)

To avoid event losses it is important for each broker to maintain two instances of its control planes ($s1$ and $s2$) during the reconfiguration process. An event published during the reconfiguration is forwarded by the data plane and tagged with the version of the control plane used for table lookup so that its next hop can use the same version to ensure consistency. Moreover, any broker will not activate $s2$ before $s2$ become convergence at all brokers (i.e. STEP4). This way the correctness of event notification is ensured. To cope with uncertain delays in the network, each step of the protocol is enforced by atomic operation [17]. That is, next step will not be triggered unless all brokers have acknowledged accomplishment of the current step.

6. PERFORMANCE EVALUATION

To evaluate the performance of DYNATOPS, we created two models that emulate the real world subscription dynamics, and compared DYNATOPS with several well-known topic-based pub/sub implementations for its subscription management and event publication efficiencies.

6.1 Dynamic Subscriptions Modeling

We considered two models of subscription changes: (1) a location-based subscription model that emulates users' dynamic subscriptions in many geosocial networking applications and location-based services; and (2) a generic Poisson dynamic subscription model that emulates changing interests of users in timely and popular topics.

6.1.1 Location-based Subscription Model

To create a large-scaled dynamic subscription model, we considered a twitter dataset [44] containing 3 million location checkins (in longitude and latitude coordinates) from over 40K Twitter users in the U.S. for 3 months period from Aug. 1st 2010 to Oct. 31st 2010. To convert the dynamic location checkins into dynamic subscriptions, we divided the U.S. geography into grids of one degree of latitude by one degree of longitude, the size of which is about 3500 $mile^2$. This results in 20×60 (i.e. 1200) grids, and we considered each of them as a location topic (i.e. totally 1200 topics). Users' checkins over time at different grids are considered as traces of their movement.

We presented our preliminary findings on users' grid checkins in Fig. 5a and 5b. We observed over 450K grid visits events (we treat users' successive checkins to locations in the same grid as a single event), from which we extracted over 160K unique (user,grid) pairs. 1/3 of them are single-time visit (i.e. a user visits to the grid only once) while the other 2/3 are at least repeated once by the user (Fig. 5a). We also analyzed the linger time of all the events. We treated the lapse of time before a user checked into a new grid as the linger time he/she stayed in the current grid. Our results indicates that about half of the 450K visits have a linger time less than a day. On the other hand, there are 10% of the visits are relatively long-lived, with a linger time over a week (Fig. 5b).

For users' dynamic subscriptions, we assume a user always subscribed to the grids he/she was residing in. Furthermore, during experiments we let each user randomly choose 4 other users as friends, and constantly follow/subscribe to the current grids/locations of his/her friends(Fig. 5c). This mimics a geosocial networking application, e.g. foursquare, where users can share their locations and activities with friends.

To experiment the pub/sub system, we considered 1200 brokers such that each broker is located inside a grid. The RTT delay between a pair of brokers and that between a broker and a user are random variables with their means proportional to their geographical distances.

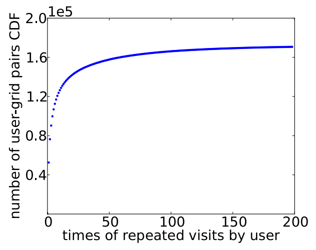
6.1.2 Generic Poisson Subscription Model

In this model, we emulated a time period of $T = 100hrs$ with time granularity of $\Delta t = 1hr$. We experimented with 50K users, 100 brokers and 1000 topics. Each user subscribes to 5 topics according to the patterns of their subscriptions. The dynamics of subscription changes over time is modeled as a Poisson process. We considered three patterns for users' topic subscriptions:

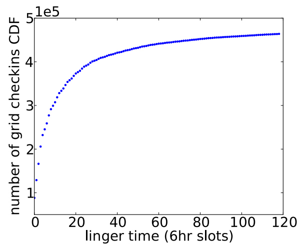
- Uniform Distribution: users subscribe to topics from the topic space in a uniform random manner.
- Zipf Distribution: topic popularity follows Zipf distribution. The probability for the i^{th} topic in the topic space is proportional to $(i+1)^{-\sigma}$, $i = 1, 2, \ldots, |P|$.
- Multimodal: users' subscriptions fall into modes. We evenly partitioned the topic space into $m = 100$ modes, and each mode contains $n_{mode} = 10$ topics. A user first randomly select a interested modes and then choose topics uniformly at random from the selected mode.

6.2 Comparison Systems

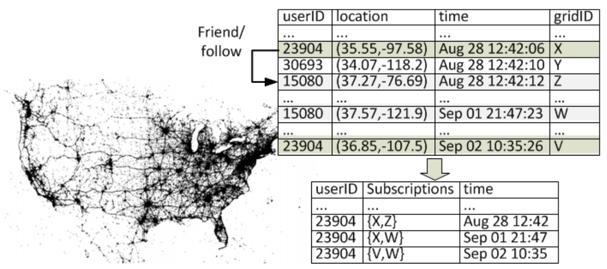
We evaluated DYNATOPS along multiple dimensions by extensive simulations. The key dimensions that serve as



(a) distribution of users' visited grids in number of repeated visits



(b) distribution of users' grid-visits in their linger time



userID	location	time	gridID
...	...	...	...
23904	(35.55,-97.58)	Aug 28 12:42:06	X
30693	(34.07,-118.2)	Aug 28 12:42:10	Y
15080	(37.27,-76.69)	Aug 28 12:42:12	Z
...	...	...	...
15080	(37.57,-121.9)	Sep 01 21:47:23	W
...	...	...	...
23904	(36.85,-107.5)	Sep 02 10:35:26	V

userID	Subscriptions	time
...	...	...
23904	{X,Z}	Aug 28 12:42
23904	{X,W}	Sep 01 21:47
23904	{V,W}	Sep 02 10:35

(c) Location-based dynamic subscriptions from Twitter data

Figure 5: dynamic subscription model from Twitter dataset

metrics for our study include *subscription management overhead*, as the number of control messages exchanged between brokers to update topic trees when brokers' subscription changes, and to update information in clusters; *notification delay* as the average latency for publications to be delivered from publishers to subscribers; *notification overhead* as the number of messages to deliver event publications; and *reconfiguration overhead* as the number of control messages for the reconfiguration protocol.

We compared DYNATOPS with several existing topic-based pub/sub systems of different categories: 1) *Bayeux*, a well-known pub/sub system atop the Tapestry DHT [10]; Bayeux was picked because it is a rendezvous-based pub/sub system on a structured overlay – providing a common ground for comparing with our system. Unlike DYNATOPS, Bayeux does not optimize its brokers based on subscriptions or perform reconfigurations, giving us an opportunity to test the value of these techniques. 2) *Topic Connected Overlay(TCO)*, a pub/sub overlay [19, 33, 11, 12] that eliminates unrelated relay brokers to provide optimal notification overhead efficiency by connecting brokers that subscribe to the same topic to form a connected subgraph; Since TCO optimizes broker topology at all times for notification efficiency, it again gives us an opportunity to test the performance and efficiency of our cost-driven broker reconfiguration policy. 3) *GeoPS* [26], a pub/sub service specific to location-based subscriptions. It takes advantage of geography-aware overlay hierarchy and geocasting technique for efficient subscription management and publication notifications. Furthermore, to study the value of the proposed schemes, we considered three versions of DYNATOPS system: (a) DYNATOPS(BNR) explores the benefit of the broker network reconfiguration scheme with an equivalent user placement scheme as other systems; (b) DYNATOPS(UP) explores the benefit of the user placement scheme only, and (c) DYNATOPS is the overall mechanism that incorporates both user placement and broker network reconfiguration schemes.

We implemented our simulator and all the above systems in Java. In our simulation, the broker networks of Bayeux and GeoPS were considered to be static. For the Bayeux simulation, brokers join the Tapestry Overlay with random Ids uniformly distributed over the Id space. For the GeoPS simulation, we divided the geography into power-of-2 grids on both edges of a rectangular geography as required by the GeoPS system. Specifically, we divide the U.S. into 32×32 grids (i.e. 1024 in total). TCO requires its broker overlay to be reconfigured whenever topic-connected property is no longer hold due to subscription changes on brokers. We reconfigure TCO by running the DCB-M algorithm [12] for partitions of nodes having changed subscriptions. Since there is no existing reconfiguration protocol to refer to, we assume the message overhead to conduct each reconfiguration is the number of links that are changed in the network topology. Users are placed on brokers at the start of each simulation, and their subscriptions change over time following the specific subscription model. Since Bayeux and TCO do not have a specific user placement policy, we considered two commonly used policies in the simulation: 1) Static selection, where a user is statically assigned a broker and 2) Location-based selection, where each broker is responsible for users in a specific region and users handover to new brokers when they move to different regions.

6.3 Experimental Results

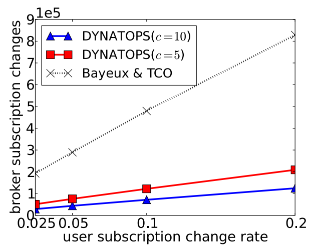
6.3.1 Basic Results

We experimented DYNATOPS and compared its performance with existing systems under the two subscription models.

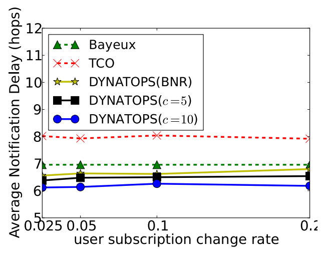
location-based subscription model: Figure 6 shows the results for the location-based subscription model. We compared DYNATOPS with Bayeux and GeoPS in their subscription and publication performances. For Bayeux, we considered both static and location-based user placement policies indicated by "Bayeux(static)" and "Bayeux(loc)". Furthermore, we considered proximity neighbor selection(PNS) in its overlay construction, along with location-based user placement, indicated by "Bayeux(loc+PNS)". When experimented with user placement technique, we formed broker clusters (see Section 4) based on their geographical proximity to avoid extensive maintenance overhead, and let each cluster manage users in a continuous geographical area close to it.

We observe that by grouping users with similar subscriptions, DYNATOPS significantly reduces the subscription management overhead against other systems (Fig. 6a). Moreover, we observe that the overhead first decreases with the increase of the cluster size c, which indicates the reduction in topic tree updates in the broker network. Under large cluster sizes (e.g. $c = 50$), however, the overhead for updating brokers' states in a cluster becomes significant, so the total subscription management overhead starts to increase.

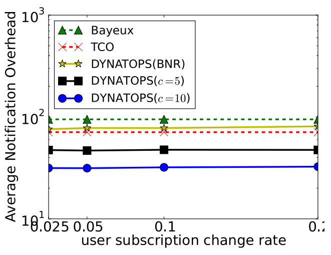
Fig. 6b shows the standard deviation in subscription load on brokers. We observe that pure similarity based user placement ($w_l = 0$) worsens the load imbalance on brokers, especially when the cluster size is large. However, the is-



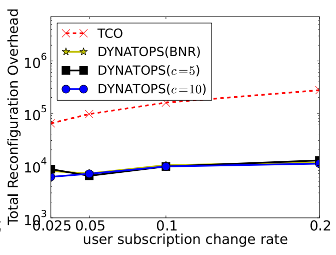
(a) brokers' subscription changes



(b) notification delay

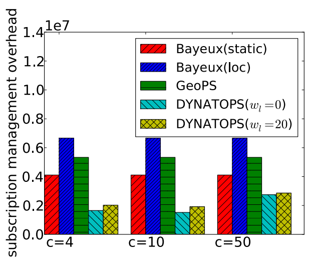


(c) notification overhead

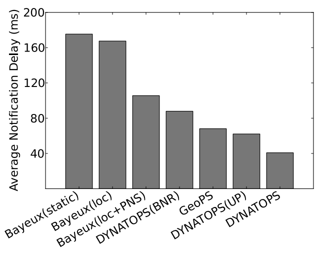


(d) reconfiguration overhead

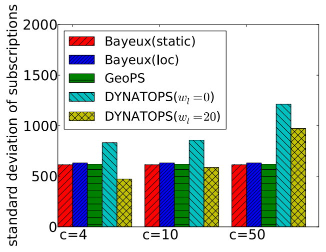
Figure 7: Poisson subscription model results under varying rate of user subscription changes with multimodal pattern



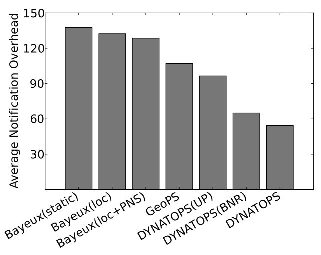
(a) subscription management overhead ($T_{update} = 12hrs$)



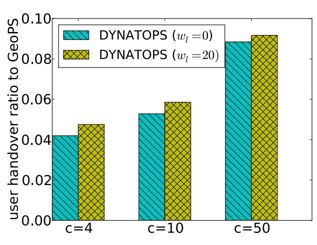
(d) notification delay ($T_{free} = 12hrs$, $c = 10$, $w_d = 1$)



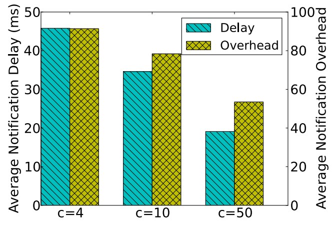
(b) standard deviation of brokers' subscription load ($T_{update} = 12hrs$)



(e) notification overhead ($T_{free} = 12hrs$, $c = 10$, $w_d = 1$)



(c) ratio of users' handovers in DYNATOPS to that of GeoPS

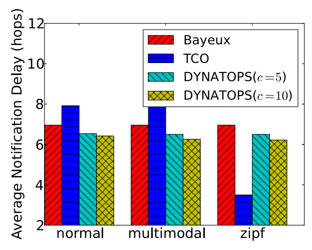


(f) DYNATOPS notification performance

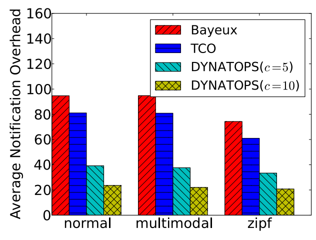
Figure 6: location-based subscription model results

sue is greatly improved by adjusting the weight for the load factor in the algorithm. We also evaluated the number of user handovers due to mobility in GeoPS and DYNATOPS. We observe that DYNATOPS reduces the user handovers by over 80% against GeoPS as shown in Fig. 6c.

Both user placement and broker network reconfiguration can improve notification delay (Fig. 6d) and overhead (Fig. 6e). This is because by clustering users with the same topic the user placement can reduce the number of brokers that need to subscribe to the topic so as to reduce the size of the topic routing tree. Combining the two algorithms DYNATOPS is highly efficient and it provides over 60% improvement against Bayeux and 30% against GeoPS in delay, and over 40% improvement against Bayeux and 20% against GeoPS in overhead (DYNATOPS $w_d = 1$). Furthermore, the notification performance improvement increases as the increase of the cluster size (Fig. 6f). This is because with larger cluster size the user placement makes the brokers' subscriptions more skewed, which favors the broker network reconfiguration to reduce unrelated relays in the topology.



(a) notification delay



(b) notification overhead

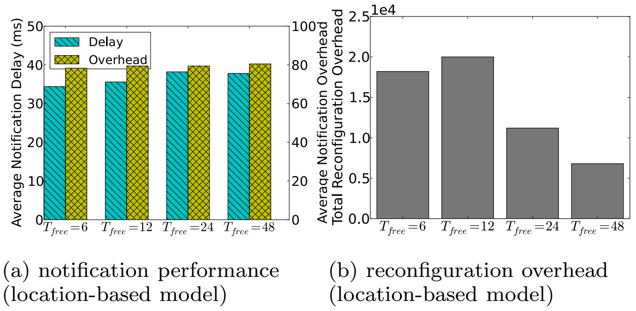
Figure 8: Poisson subscription model results in different patterns
(subscription change rate = 0.1, $T_{free} = 10hrs$)

Poisson subscription model: Fig. 7 shows the results of the Poisson subscription model where users subscriptions change with varying Poisson rates. We compared DYNATOPS with Bayeux and TCO for subscription and publication performances. In both Bayeux and TCO, we assume users are statically assigned brokers in a uniformly random manner. For publication performances, we also considered DYNATOPS(BNR) where the user placement is the same as those for Bayeux and TCO, to evaluate DYNATOPS under the broker network reconfiguration technique along. For simplicity, in the experiment we assumed a unit RTT delay between any pair of brokers so that the notification delay is dominated by the number of overlay hops.

Fig. 7a to 7d show the experimental results where users subscriptions follow the multimodal pattern under varying Poisson rate of user subscription changes. We observe that DYNATOPS reduces the number of brokers' subscription changes by 80% against other systems (Fig. 7a), resulting in significantly less subscription management overhead. Fig. 7b and 7c show the publication delay and overhead performances of different pub/sub systems. TCO achieves a better overhead performance than Bayeux and DYNATOPS(BNR) because of the topic-connected property. However, we observe a worse delay performance because its suboverlay construction is not delay aware. On the other hand, DYNATOPS considering both user placement and broker network reconfiguration outperforms Bayeux and TCO on both delay and overhead. It provides 10% improvement in delay and over 50% improvement in overhead against the compared systems.

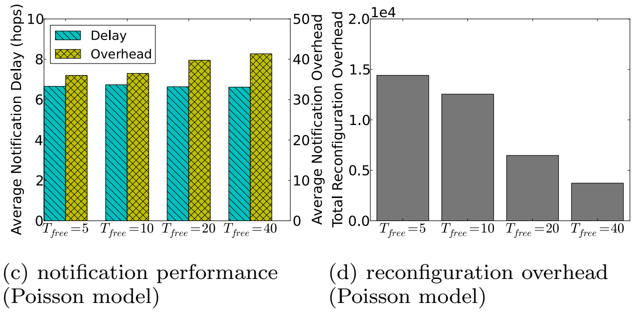
We also compared the reconfiguration cost between DYNATOPS and TCO (Fig. 7d). The reconfiguration overhead for TCO increases dramatically with the increase of the rate of users subscription changes. This makes the scheme infeasible to be applied in highly dynamic environment. On the other hand, DYNATOPS's reconfiguration overhead is less sensitive to the rate of users subscription changes. It is over 80% less than that of TCO when users' subscriptions change fast.

The notification performance of each system under different subscription patterns of users are shown in Fig. 8a and Fig. 8b. We observe that TCO has a low delay under zipf pattern where subscription is highly skewed but worse under other subscription patterns. On the other hand, DYNATOPS outperforms other systems under all subscription patterns.



(a) notification performance (location-based model)

(b) reconfiguration overhead (location-based model)



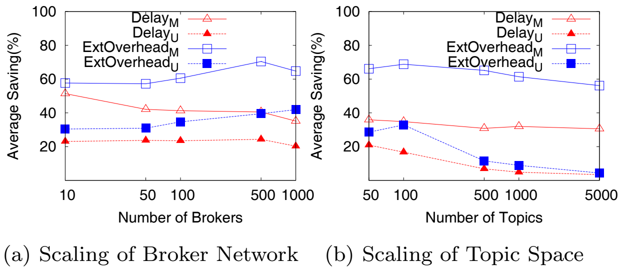
(c) notification performance (Poisson model)

(d) reconfiguration overhead (Poisson model)

Figure 9: results under varying reconfiguration free period. For location-based model, $c = 10$; For Poisson model, subscription change rate = 0.025 and $c = 5$.

6.3.2 *Reconfiguration free period*

To gain better understanding of the performance, we experimented with various reconfiguration free period T_{free} in both location-based and Poisson models. In Fig. 9. We observe that with the increase of the reconfiguration free period, the notification performance degrades slightly and the reconfiguration overhead decreases because less reconfigurations were triggered. It is worth noting that DYNATOPS only experienced a slight degradation in notification efficiency when the reconfiguration free period increases. This is because the user placement technique stabilized brokers subscriptions such that they do not experience dramatic subscription changes over time in spite of dynamic users' subscriptions.



(a) Scaling of Broker Network (b) Scaling of Topic Space

Figure 10: Performance savings of DYNATOPS configurations against CHash under various size of broker network $|B|$ and of topic space $|P|$. We varied $|B|$ with fixed $|P| = 100$, and varied $|P|$ with fixed $|B| = 100$. "M" denotes multimodal pattern and "U" denotes uniform pattern.

	varying $\lvert B\rvert$			**varying $\lvert P\rvert$**		
size	**10**	**100**	**1000**	**50**	**500**	**5000**
time	28ms	184ms	96s	98ms	502ms	1.9s

Table 1: The computation time of the DYNATOPS-Greedy algorithm under various size of broker network $|B|$ and of topic space $|P|$. For varying $|B|$, we fixed $|P| = 100$, and for varying $|P|$ we fixed $|B| = 100$.

6.3.3 *Scalability*

We also experimented with various sizes of the broker network and of the topic space to evaluate the time efficiency and performance of the DYNATOPS-Greedy configuration algorithm. We ran the algorithm on a Dell workstation with a QuadCore 2GHz CPU and 2G memory. The performance of the output DYNATOPS configurations are compared against that of a bootstrap configuration from consistent hashing of brokers public keys or IP addresses for their nodeIDs. This is the configuration approach adopted by most existing DHT-based pub/sub systems [30, 40, 41, 15]. The performance was evaluated under two subscription patterns on brokers: uniform distribution subscriptions and multimodal subscriptions. Figure 10 and Table 1 show the performance and computation time of the algorithm. We observe that DYNATOPS configuration always provides improved notification performance against consistent hashing under various size of broker networks and topic spaces. The improvement is larger under skewed subscription patterns than the uniform pattern. Furthermore, the proposed configuration algorithm is efficient to compute DYNATOPS configurations for a large broker network and topic space.

7. DISCUSSION AND CONCLUSION

In this paper, we propose and develop DYNATOPS, a pub sub system for societal scale applications, that can deal with dynamic, yet short lived subscriptions. DYNATOPS users are moderately repositioned on brokers for efficient subscription management and brokers are moderately repositioned on the overlay structure for efficient event notifications, to adapt to the publications and subscription dynamics. Unlike existing systems where the overlay topology changes in a self-organizing manner in response to the changes of subscriptions, DYNATOPS performs planned reconfiguration utilizing a cost-driven reconfiguration process. The proposed approach can significantly reduce the reconfiguration cost while maintaining a high notification performance as compared to state-of-the-art systems.

The centralized reconfiguration requires the participation of all brokers in the structured overlay. To mitigate the scalability concern on the reconfiguration computation and protocol when the network size grows, a hybrid pub/sub network similar to [18] which exploit both unstructured clustering of similar peers and structured rendezvous routing may be adopted. In the hybrid network, only gateway brokers from each cluster of gossip-based unstructured overlays will join the core structured rendezvous network and participate into the structure reconfiguration. The performance of the hybrid structure will be investigated in the future work. We will also extend the centralized configuration manager to a distributed implementation, mitigating any reliability concerns.

We have implemented a prototype of the DNATOPS broker system atop a Chord DHT implementation OpenChord [3]. We will next deploy this broker network on a campus cluster and evaluate the system on an emulated Internet using Modelnet [2]. We are also extending DYNATOPS to implement a large scale mobile alerting system that exploits the geographical and societal correlations inherent in societal scale notification systems.

8. REFERENCES

[1] Farecast. http://www.ics.uci.edu/~dsm/papers/farecast_techreport.pdf.

[2] Modelnet. http://issg.cs.duke.edu/modelnet.html.

[3] Openchord. `http://sourceforge.net/projects/open-chord/`.

[4] techreport. `http://www.ics.uci.edu/~yez/dynatops_tech.pdf`.

[5] D. Rosenblum A. Carzaniga and A. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *PODC*, 2000.

[6] D. S. Rosenblum A. Carzaniga and A. L. Wolf. Design and evaluation of a wide-area event notification service. In *TOCS*, 2001.

[7] Y. Cheung A. King and H. Jacobsen. Publisher placement algorithms in content-based publish/subscribe. In *ICDCS*, 2010.

[8] I. Aekaterinidis and P. Triantafillou. Pastrystrings: A comprehensive content-based publish/subscribe dht network. In *ICDCS*, 2006.

[9] et al. B. F. Cooper. Pnuts: Yahoo!s hosted data serving platform. In *VLDB Endow.*, 2008.

[10] et al B. Y. Zhao. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. In *UCBerkeley Tech. Rep.*, 2001.

[11] H. Jacobsen C. Chen and R. Vitenberg. Divide and conquer algorithms for publish/subscribe overlay design. In *ICDCS*, 2010.

[12] R. Vitenberg C. Chen and H. Jacobsen. Scaling construction of low fan-out overlays for topic-based publish/subscribe. In *ICDCS*, 2011.

[13] F. Cao and J. P. Singh. Efficient event routing in content-based publish/subscribe service network. In *INFOCOM*, 2004.

[14] F. Cao and J. P. Singh. Medym: Match-early with dynamic multicast for content-based publish-subscribe networks. In *Middleware*, 2005.

[15] A. Post D. Sandler, A. Mislove and P. Druschel. Feedtree: Sharing web micronews with peer-to-peer event notification. In *IPTPS*, 2005.

[16] G. Li et al. Adaptive content-based routing in general overlay topologies. In *Middleware*, 2008.

[17] R. Strong F. Cristian, H. Aghili and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation*, 1995.

[18] A. H. Payberah F. Rahimian, S. Girdzijauskas and S. Haridi. Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks. In *IPDPS*, 2011.

[19] et al G. Chockler. Constructing scalable overlays for pub-sub with many topics: Problems, algorithms and evaluation. In *PODC*, 2007.

[20] et al G. Chockler. Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS*, 2007.

[21] V. Muthusamy G. Li and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. In *Middleware*, 2008.

[22] N. Venkatasubramanian H. Jafarpour, S. Mehrotra and M. Montanari. Mics: An efficient content space representation model for publish/subscribe systems. In *DEBS*, 2009.

[23] et al H. Liu. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *IMC*, 2005.

[24] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wireless Networks*, 2004.

[25] et al I. Stoica. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.

[26] U. Lee J. H. Ahnn and H. J. Moon. Geoserv: A distributed urban sensing platform. In *CCGrid*, 2011.

[27] R. S. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe service. In *SRDS*, 2009.

[28] C. Jayalath K.R. Jayaram and P. Eugster. Parametric subscriptions for content-based publish/subscribe networks. In *Middleware*, 2010.

[29] O. Kasten L. Fiege, F. C. Gartner and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *MIDDLEWARE*, 2003.

[30] et al M. Castro. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *PJSAC*, 2002.

[31] R. Oliveira M. Matos, A. Nunes and J. Pereira. Stan: exploiting shared interests without disclosing them in gossip-based publish/subscribe. In *IPTPS*, 2010.

[32] G. Muhl. Large-scale content-based publish/subscribe systems. In *PhD thesis*, Darmstadt Univ. of Technology, 2002.

[33] M. Onus and A. W. Richa. Minimum maximum degree publish-subscribe overlay network design. In *INFOCOM*, 2009.

[34] R. Guerraoui P. Th. Eugster, P. A. Felber and A. Kermarrec. The many faces of publish/subscribe. In *ACM Computing Surveys (CSUR)*, 2003.

[35] P. R. Pietzuch and J. M. Bacon. Hermes: A distributed event-based middleware architecture. In *DCSW*, 2002.

[36] et al R. Baldoni. Tera: topic-based event routing for peer-to-peer architectures. In *DEBS*, 2007.

[37] L. Querzoni R. Baldoni, R. Beraldi and A. Virgillito. Efficient publish/subscribe through a self-organizing broker overlay and its application to siena, 2007.

[38] et al. S. Girdzijauskas, G. Chockler. Magnet: practical subscription clustering for internet-scale publish/subscribe. In *DEBS*, 2010.

[39] et al S. Voulgaris. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *IPTPS*, 2006.

[40] et al S.Q. Zhuang. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, 2001.

[41] R. Peterson V. Ramasubramanian and E. G. Sirer. Corona: A high performance publish-subscribe system for the world wide web. In *NSDI*, 2006.

[42] R. Vitenberg V. Setty, M. Steen and S. Voulgaris. Poldercast: Fast, robust and scalable architecture for p2p topic-based pub/sub. In *Middleware*, 2012.

[43] V. Muthusamy Y. Yoon and H. Jacobsen. Foundations for highly available content-based publish/subscribe overlays. In *ICDCS*, 2011.

[44] K. Lee Z. Cheng, J. Caverlee and D. Z. Sui. Exploring millions of footprints in location sharing services. In *ICWSM*, 2011.