Publish/Subscribe Service in CAN-based P2P Networks: Dimension Mismatch and The Random Projection Approach

Duc A. Tran
Department of Computer Science
University of Massachusetts, Boston, MA 02125
Email: duc@cs.umb.edu

Thinh Nguyen
Department of Electrical and Computer Engineering
Oregon State University, Corvallis, OR 97331
Email: thinhq@eecs.oregonstate.edu

***Abstract*—CAN is a well-known DHT technique for content-based P2P networks, where each node is assigned a zone in a virtual coordinate space to store the index of the data hashed into this zone. The dimension of this space is usually lower than the data dimension, thus we have the problem of dimension mismatch. This problem is widely addressed in the context of data retrieval that follows the traditional request/response model. However, little has been done for the publish/subscribe model, which is the focus of our paper. We show that dimension mismatch in CAN-based publish/subscribe applications poses new challenges. We furthermore investigate how a random projection approach can help reduce the negative effects of dimension mismatch. Our theoretical findings are complemented by a simulation-based evaluation.**

***Index Terms*—P2P, publish/subscribe, random projection, CAN, DHT**

I. Introduction

Search applications can be categorized into two models: request/response and publish/subscribe (in short, pub/sub). In the former, a search query is submitted on demand expecting the results to return immediately; if they do not exist, a response indicating so is returned. Contrarily, a query in the pub/sub model is submitted and stored in advance. The results may already exist, which will be returned immediately; otherwise, the query subscriber will be notified when the matching results later become available. Thus, a main problem for request/response search systems is to manage existing data in advance for fast data retrieval at a later time, while the corresponding problem for pub/sub systems is to manage the queries in advance for fast query matching in the future.

We are interested in the pub/sub service deployed in a P2P network. We particularly work on the case the network overlay is structured according to the CAN technique [1], one of the well-known Distributed Hash Tables (DHTs) designs [1]–[4] for self-organizing and scalable P2P networks. In CAN, each node is given a unique identifier that represents a non-overlapping rectangular zone partitioned from a virtual multi-dimension space, called the CAN space. For routing purposes, each node has a list of neighbor nodes whose zones are adjacent to its zone. Consequently, the number of neighbors per node is proportional to the dimension of the CAN space, which is independent from the network size (i.e., the number of nodes in the network). Although the other DHT techniques have a node overhead increasing with the network size, CAN is favored over them only when the CAN dimension is low.

The motivation behind CAN is distributed search applications that follow the request/response model. In the indexing phase, each data object is hashed into a point in the CAN space and its index is stored at the node whose zone contains this hash point. Thus, when a query for an object is initiated, we route the query to the node owning the query's hash point and search there locally for the object.

Deploying a pub/sub service on top of CAN is not as straightforward. From the database perspective, because we typically model a data object as a point and a query as a range of points, we need to address the range indexing problem in pub/sub systems, which is more challenging than the point indexing problem as in request/response systems.

From the networking perspective, due to its range, a subscription query may be replicated at multiple nodes to wait for notification of all possible matching data objects. Hence, the number of subscriptions stored in the network can be large, resulting in not only the communication cost to replicate the subscriptions, but also high storage cost for each node and long time to match an object against a subscription query. We need to minimize unnecessary replications, yet at the same time store the queries in the network intelligently so that data notification remains efficient. Such issues are not present in request/response systems.

Because of the low dimensionality of the CAN space, another challenge to a CAN-based pub/sub system is due to the mismatch between the CAN dimension and the data dimension. Data can be, and usually, of high dimension, such as in applications searching documents, multimedia, and sensor data, which normally are associated with many attributes. It is difficult to hash similar high-dimension data objects into zones in a low-dimension space which are close to each other, making the search for a continuous range of data highly inefficient. For the request/response model, in which data need to be preprocessed and queries are subsequent and on-demand, dimension mismatch can be resolved effectively

by locality-sensitive dimensionality reduction techniques such as Locality Sensitive Hashing [5], Latent Semantic Indexing [6], and Random Projection [7].

For the pub/sub model, because subscription queries are submitted in advance, we need to store similar subscriptions in nodes of close proximities so as to make the data notification process efficient. While the aforementioned reduction techniques apply only to singular points whose similarity can be defined by a distance metric, this metric is not applicable to queries which are ranges of points.

In this paper, we explore the application of Random Projection in pub/sub networks that are specifically built on CAN. We investigate how Random Projection is used to map the pub/sub data and queries into a CAN space of lower dimension. We also propose a strategy aimed to avoid replicating subscriptions, guaranteeing that any query is matched with all possible data objects using an efficient notification process.

Next, we discuss the related work in Section II. We present some preliminary information in Section III, then the random projection approach in Section IV. Simulation results are reported in Section V. The paper is concluded in Section VI.

II. Related Work

The simplest architecture for a distributed pub/sub network is the broadcast approach, in which a subscription traverses a broadcast tree to reach all the nodes. This approach is not cost-effective. A much better option is to replicate a subscription in a set of select nodes where satisfying data may likely be sent to. Most techniques of this option employ a Distributed Hash Table (DHT) [1]–[4]. A DHT is used to send a subscription query or data object to a node that is the result of the hash function. The goal is that the node storing a subscription and that receiving a satisfactory data object are either identical or within a proximity of each other. Scribe [8] uses Pastry [3] to map a subscription to a node based on topic hashing, thus those subscriptions and data objects with the same topic are mapped to the same node. Instead of Pastry, the CAN DHT [1] and the Chord DHT [2] structures are employed in Meghdoot [9] and [10], respectively. A technique that can be used atop any such DHT structure was proposed in [11]. Non-DHT techniques also exist, such as Sub-2-Sub [12] and R-tree-based [13].

Meghdoot [9] is the only existing CAN-based pub/sub technique that works for multi-dimensional data space. In Meghdoot, each query in d dimensions is mapped a point in $2d$ dimensions. Therefore, the P2P network is virtualized in a CAN space of $2d$ dimensions. Our work is different. First, our mapping is based on random projection. Second, our purpose is to reduce the data dimension to a lower CAN dimension. Lastly, while the design of the CAN network in Meghdoot is built dependently on the pub/sub data, we can work with the case that the CAN network already exists on which we build the pub/sub service later.

III. Preliminaries

In CAN, each node V is virtualized as a point at location $Point(V)$ owning a zone $Zone(V)$ in a k-dimension unit cube $\mathcal{C} = [0,1]^k$. For a network of n nodes $\{V_1, V_2, ..., V_n\}$, ranked in the time order they join the network, its construction is explained recursively as follows:

- Case $n = 1$, i.e., there is only one node V_1 in the network $Point(V_1) = \underbrace{(1/2, 1/2, ..., 1/2)}_{k}$ and $Zone(V_1) = \mathcal{C}$
- Case $n > 1$: $n-1$ nodes $\{V_1, V_2, ..., V_{n-1}\}$ already assigned their location and zone, the corresponding assignment for node V_n is determined as follows:
  1) Choose a point $p \in \mathcal{C}$ uniformly in random
  2) Let V_i be the node such that $p \in Zone(V_i)$. Suppose that j is the dimension of this zone's largest side $[min_j, max_j]$
  3) Halve $Zone(V_i)$ into Z_1 and Z_2 that share every side with $Zone(V_i)$ except that the j^{th}-dimension side of Z_1 is $[min_j, (min_j + max_j)/2]$ and that of Z_2 is $[(min_j + max_j)/2, max_j]$
  4) Node V_i will be reassigned to new zone Z_1 and located at the center of this zone
  5) Node V_n will be assigned to zone Z_2 and located at the center of this zone

For routing in CAN, each node V is required to maintain a list of neighbor nodes that own zones adjacent to $Zone(V)$. Routing from one zone to another is by relaying via neighbor nodes, greedily getting as close geometrically to the destination as possible. The routing and node overhead costs of CAN are therefore $O(n^{1/k})$ and $O(k)$, respectively. In particular, the neighborhood size of a node in CAN is at least $2k$, making CAN potentially favorable over other DHT techniques (whose neighborhood size is typically $\log(n)$) only when $2k \leq \log(n)$.

Search services can be implemented on CAN. Suppose that the data space $\mathcal{D}$ is d dimensional; hence, each data object is represented as a d-tuple. CAN works best for exact-match lookup. Given a data object with value $x \in \mathcal{D}$, we hash it to a point in the CAN space $h(x) \in \mathcal{C}$ and store the index of x at the node whose zone contains $h(x)$. A query searching for a data value x can be easily answered by routing to the zone containing the hash point $h(x)$ of the query value.

For pub/sub applications on CAN, we need to store and index subscription queries rather than data objects. A subscription query is usually specified as a region of points in $\mathcal{D}$. We assume that each query q is a sphere. Simple to express, spherical queries are more convenient than queries of other types when the data space is high dimensional. We need to find two mapping methods, h_s and h_o, so that (1) given a subscription query q, it will be stored at the set of nodes $h_s(q)$; and (2) given a data object x, it will be advertised at the set of nodes $h_o(x)$. The following requirements are important. These two mappings should guarantee that if x satisfies q an advertisement of x must reach at least a node that stores q; i.e., $h_o(x) \cap h_s(q) \neq \emptyset$. The sets $h_s(q)$ and $h_o(x)$ should be small to keep the cost of storage and communication low. For search efficiency, subscription locality and data locality should also be preserved. In other words, similar data objects should

be mapped to similar sets of nodes and so should similar subscription queries.

If the data dimension d equals the CAN dimension k, it is easy to find h_s and h_o because no dimensionality reduction is needed. For example, first, we can use a linear scaling to fit the data space into the CAN space. Then, for each subscription query q, we find the node containing the center point of q and conduct an expanded search in the surrounding nodes to find the other nodes whose zones intersect with q; query q will be stored at those visited nodes. When a data object is published, its publication is advertised to the node whose zone contains point x. It is more challenging, though, when d is larger than k because of the reasons explained in Section I. We propose our approach in the next section.

IV. The Random Projection Approach

We base our solution on Random Projection. Inspired originally by Johnson-Lindenstrauss Lemma [7] and then by its subsequent related work (e.g., [14]–[16]), projection onto $k << d$ random dimensions can be applied on a d-dimension point set such that the distance between two points after the projection remains within a small constant factor of the original distance. Let $\{\overrightarrow{u_1}, \overrightarrow{u_2}, ..., \overrightarrow{u_k}\}$ be k random d-dimension orthonormal vectors. Because, for high dimensions d and k, generating these vectors can be computationally expensive, we use the random vectors as recommended in [14]: they are chosen to be the column vectors of the matrix $U = \{u_{ij}\}_{d\times k}$ generated from the following distribution

$$u_{ij} = \sqrt{1/d} \times \begin{cases} +1 & \text{with prob 1/2} \\ -1 & \text{with prob 1/2} \end{cases} \quad (1)$$

Without loss of generality, suppose that the data space $\mathcal{D}$ is the d-dimension unit cube: $\mathcal{D} = [0,1]^d$. We explore two strategies of using Random Projection for CAN-based pub/sub networks: the PURE strategy and the SMART strategy. The PURE strategy is a simple way of mapping the data space into the CAN space using Random Projection. The SMART strategy is aimed at avoid replicating the subscriptions by taking into account their overlapping relationships.

A. The PURE strategy

Consider a subscription query $q = (s, r)$, which is a sphere centered at point $s \in \mathcal{D}$ with radius $r \geq 0$. Projecting this sphere on to the k random vectors, we obtain the following hyperrectangle in k dimensions: (see Figure 1)

$$(s, r) \rightarrow u(s, r) = u_1(s, r) \times u_2(s, r) \times ... \times u_k(s, r)$$

where each side $u_i(s, r) = [\langle u_i, s\rangle - r, \langle u_i, s\rangle + r]$ ($\langle\cdot,\cdot\rangle$ denotes the inner product).

This hyperrectangle may not fit in the CAN cube. To do so, we apply the following linear transformation. Suppose that r_{max} is the maximum radius possible. Letting $\alpha_i = -r_{max} + \sum_{j=1}^{d} \min(0, u_{ij})$ and $\beta_i = r_{max} + \sum_{j=1}^{d} \max(0, u_{ij})$, the following rectangle, called the *CAN-projection* of query q,

$$u_{\mathcal{C}}(q) = \prod_{i=1}^{k} \left[\frac{\langle u_i, s\rangle - r - \alpha_i}{\beta_i - \alpha_i}, \frac{\langle u_i, s\rangle + r - \alpha_i}{\beta_i - \alpha_i} \right]$$

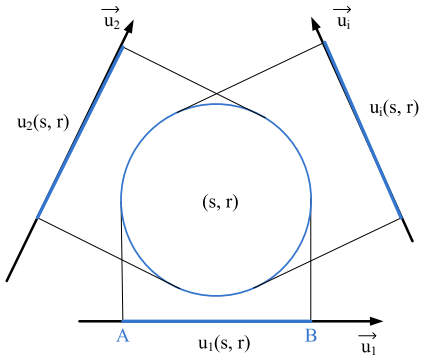


Fig. 1. Projection of a sphere onto random vectors

which is linearly transformed from $u(s, r)$, is inside the unit CAN cube. The center of this hyperrectangle is the k-dimension point

$$Center_{\mathcal{C}}(q) = \left(\frac{\langle u_1, s\rangle - \alpha_i}{\beta_i - \alpha_i}, \frac{\langle u_2, s\rangle - \alpha_i}{\beta_i - \alpha_i}, ..., \frac{\langle u_k, s\rangle - \alpha_i}{\beta_i - \alpha_i} \right)$$

1) Query Subscription: Our subscription strategy is to store the query q in the following set of nodes whose zone intersects with q's CAN-projection:

$$h_s(q) = \{node\ V_i \mid Zone(V_i) \cap u_{\mathcal{C}}(q) \neq \emptyset\}$$

To implement this strategy in the CAN network, we follow the protocol below:

1) Use the CAN routing protocol to send q to the node V_q such that $Zone(V_q)$ contains $Center_{\mathcal{C}}(q)$.
2) For each node V that receives q, forward it to each neighbor node V' such that $Zone(V') \cap u_{\mathcal{C}}(q) \neq \emptyset$. Node V' follows the same procedure as V does.

2) Data Notification: Using the same random projection method explained earlier, each data object $x \in \mathcal{D}$ is mapped to the following point in the CAN cube (imagine x as a zero-radius query $(x, 0)$):

$$x \rightarrow u_{\mathcal{C}}(x) = \left(\frac{\langle u_1, x\rangle - \alpha_i}{\beta_i - \alpha_i}, \frac{\langle u_2, x\rangle - \alpha_i}{\beta_i - \alpha_i}, ..., \frac{\langle u_k, x\rangle - \alpha_i}{\beta_i - \alpha_i} \right)$$

When a data object $x \in \mathcal{D}$ becomes available, using CAN routing, we advertise it to the node V_x such that $Zone(V_x)$ contains the point $u_{\mathcal{C}}(x)$; i.e., $h_o(x) = \{V_x\}$.

It is obvious that if x satisfies a query q, then $u_{\mathcal{C}}(x) \in u_{\mathcal{C}}(q)$. Consequently, $Zone(V_x)$ must intersect with $u_{\mathcal{C}}(q)$ and the query q must be stored at node V_x. Thus, given any data object x and subscription query q, if they match each other, they are guaranteed to always find each other.

Furthermore, given two objects x and y, their similarity is retained in the CAN space. Indeed, denoting the Euclidean distance by $d(.,.)$, we have $d(u_{\mathcal{C}}(x), u_{\mathcal{C}}(y)) = \sqrt{\sum_{i=1}^{k} \left(\frac{\langle u_i, x-y\rangle}{\beta_i - \alpha_i} \right)^2} \leq \sqrt{\sum_{i=1}^{k} \left(\frac{\|x-y\|}{2r_{max} + \sum_{j=1}^{d} |u_{ij}|} \right)^2} = \frac{\sqrt{k}}{2r_{max}} d(x, y)$

It is also true that given two overlapping queries q and q', we have $u_{\mathcal{C}}(q \cap q') \subseteq u_{\mathcal{C}}(q) \cap u_{\mathcal{C}}(q')$. Therefore, if we define the similarity between two subscription queries by the number of common data objects, it is preserved in the CAN space.

B. The SMART Strategy

The PURE strategy allows for quick and cost-effective data notification because each data object is advertised to only one node. However, PURE may incur a large amount of subscription replicas in the network.

Since subscription queries are likely to overlap, we should take advantage of this property to minimize their replication in the network. As illustrated in Figure 3, if query q' covers query q, it must be true that $u_{\mathcal{C}}(q')$ covers $u_{\mathcal{C}}(q)$ and, consequently, $h_s(q') \supseteq h_s(q)$. In other words, if a new query is covered by an existing query, the nodes that the former query is mapped to must already store the existing query. Because those data objects that satisfy q' will be returned to notify q' anyway, which will be filtered to match q, there is no need to replicate query q further.

It should, however, be noted that as the data dimensionality increases, the number of subscription coverings is decreased because the subscription set is more sparse. As such, merely using this relationship might not be sufficiently effective in reducing the subscription load.

We propose the SMART strategy, which does not replicate a query q if an existing query q' is found such that $u_{\mathcal{C}}(q') \supset u_{\mathcal{C}}(q)$ (instead of using the condition $q' \supset q$). The likelihood of $u_{\mathcal{C}}(q') \supset u_{\mathcal{C}}(q)$ is much higher than that of $q' \supset q$, thus SMART can reduce the replication cost significantly; SMART does not replicate a query not only if it is covered by an existing query, but also if its CAN-projection is covered by the CAN-projection of an existing query.

1) Query Subscription: We associate with each query q with a node called the home node $home(q)$ – the node that stores the first copy of query q if it is replicated multiple times or the only copy otherwise. The protocol to replicate a query in the SMART strategy is as follows.

1) *Use the CAN routing protocol to send q to the node V_q such that $Zone(V_q)$ contains $Center_{\mathcal{C}}(q)$.*
2) *If there is a query q' currently stored at node V_q such that $u_{\mathcal{C}}(q') \supseteq u_{\mathcal{C}}(q)$, store query q at node $home(q')$*
3) *Else*
   a) *Set $home(q) = V_q$ and store q at V_q*
   b) *Use the PURE strategy to propagate (q, V_q) to surrounding nodes; at each node V',*
      i) *For each existing query q' stored at V' such that $u_{\mathcal{C}}(q') \subseteq u_{\mathcal{C}}(q)$ and $home(q') = V_q$, remove query q' from node V'*

An illustration on a 2-D CAN network is given in Figure 2. Queries q_1, q_2, q_3 are submitted into the network at times in that order. Query q_1 is submitted first, whose projection center $Center_{\mathcal{C}}(q_1)$ lies in the zone of node 7 and whose projection also intersects with the zones of nodes 15, 13, 11, 10, 9, 8, and 2. Therefore, q_1 is stored at these nodes and the home node of q_1 is node 7. When q_2 is submitted, it is sent to node 10 because this node's zone contains the projection center of q_2. If we use the PURE strategy, q_2 would be stored at nodes 8, 9, 10, 11. Using the SMART strategy, because node 10 already stores query q_1 and $u_{\mathcal{C}}(q_1) \supseteq u_{\mathcal{C}}(q_2)$, query q_2 will be stored at the home node of query q_1, i.e., node 7 only; hence, a significant reduction in subscription load. When query q_3 is submitted, because its projection is not covered by any other's, it is stored at nodes 2 and 3 because their zones intersect with $u_{\mathcal{C}}(q_3)$, node 2 serving as the home node of query q_3.

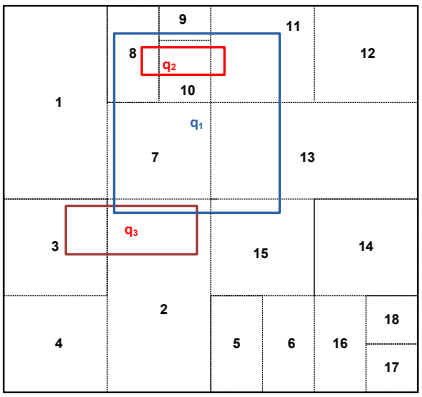


Fig. 2. Subscription replication using covering relationship: Query q_1, which is inserted first, is stored at nodes 7 (home node), 15, 13, 11, 10, 9, 8, and 2; query q_2 is stored at 7 because it is covered by q_1 whose home node is 7; query q_3, not covered by any other is stored at nodes 2 (home node) and 3

2) Data Notification: When a data object x becomes available, the data publication procedure is as follows to work with the SMART strategy:

1) *Use CAN routing to advertise it to the node V_x such as $u_{\mathcal{C}}(x) \in Zone(V_x)$*
2) *For each query q stored at node V_x such that $u_{\mathcal{C}}(x) \in u_{\mathcal{C}}(q)$, forward x to node $home(q)$ – the home node of q*
   a) *For each home node V that receives x, notify all queries that match x and call V home*

In the PURE strategy, searching node V_x is sufficient to find all the matching queries of x. In the SMART strategy, there may be subscriptions matching x, which are not stored at node V_x; those subscriptions are actually stored at the home nodes of the queries q stored at V_x such that $u_{\mathcal{C}}(x) \in u_{\mathcal{C}}(q)$. Thus, we have to visit the home nodes of such queries q to find all possible matching queries of x. For example, continuing the illustration earlier in Figure 2, suppose that a data object x satisfying query q_2 is available such that $u_{\mathcal{C}}(x)$ lies in zone 11. An advertisement will be sent to node 11. If we search node 11 alone as in the PURE strategy, we cannot find any other query beside q_1 that x satisfies. This results in a match miss because x also satisfies q_2. Using the modified publication procedure, we will forward the advertisement of x to the home node 7 of query q_1, where we will find query q_2.

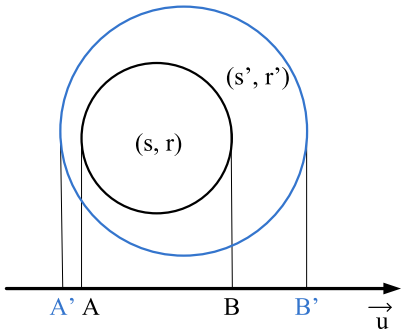


Fig. 3. Covering relationship is preserved in random projection: if a query (s', r') covers a query (s, r), the former's projection A'B' also covers the latter's AB

C. Un-Subscription

We should enable a subscription query to be unsubscribed when necessary. In the case of PURE strategy, a subscription q is removed easily by visiting all the nodes whose zone intersects with $u_{\mathcal{C}}(q)$, where q will be removed. If we use the SMART strategy, the removal of a subscription q also involves some other queries. These queries are those stored at the home node of q because their CAN-projection is covered by q's. For such a covered query q', firstly we check whether its projection is covered by that of another query other than q. If so, nothing needs to be done. Otherwise, the home node of q re-subscribes q' to the network.

V. Simulation Study

Shown in the previous section is that using our random projection approach subscription queries and their matching data objects always meet. A main impact of dimension mismatch in CAN-based pub/sub networks, which is difficult to quantify theoretically, is how it affects the subscription load and the efficiency of the notification process. We, therefore, conducted a simulation-based evaluation to study these aspects. Our evaluation was based on (1) the *subscription replication cost*: the number of replicas of each subscription submitted to the network and (2) the *event notification efficiency*: the number of nodes where the advertisement on the event that a new data object is produced is sent to find all matching subscriptions. We also studied the effect of increasing the subscription size. As discussed in Section II, we are aware of no other research on CAN-based pub/sub systems that share the same problem addressed in our paper. This study thus includes the results for our random projection approach only.

We simulated a CAN network of 1,000 nodes. For CAN to be preferable over the other DHT techniques, the CAN dimension k should be smaller than $\log(n)/2$. Thus, we considered the value of k to be in the range $\{2, 3, 4, 5\}$. We generated two sets of subscriptions, one with 10,000 subscriptions and the other 50,000 subscriptions. The centers of these subscriptions were generated uniformly in random as points in the d-dimension unit cube. The radii were chosen to a maximum of $r_{max} = 0.5$ according to two distribution models: the uniform distribution and the Pareto 80/20 distribution. The latter one is typical in practice, reflecting the case that most subscriptions are specific (i.e., small radius), only a few being extensive (i.e., large radius). We evaluated various choices for the data dimension but due to the space limit and for simplicity of conveying the insights we report the results for $d \in \{4, 8\}$. Many real pub/sub applications, such as for watching stock markets and monitoring weather, have this range of dimensionality.

A. Subscription Replication Cost

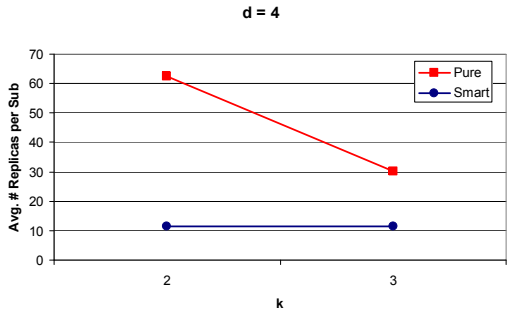
When a subscription is submitted, PURE replicates it at every node whose zone intersects with the subscription's projection. SMART aims to avoid this replication using the covering relationship between subscription projections. Thus, SMART certainly results in less subscription load in the network. The load reduction is substantial as shown in Figure 4. It is more obvious when k is smaller no matter the query model. For example, when $k = 2$ and the uniform query model is used, while PURE results in 64 replicas/subscription, SMART replicates a subscription only 12 times; hence, 80% saving (Figure 4(a)). The closest case is when $k = 5$, $d = 8$; yet, a 30% saving in subscription load is obtained with SMART (Figure 4(c) and Figure 4(d)). In any configuration, a subscription in SMART is replicated at no more than 12 nodes, or a 1.2% of the network size. We should note that the load in the Pareto cases is much smaller than in the uniform cases. This is because with Pareto, most of the subscriptions queries are highly specific (i.e., small radius), thus their projections do not intersect with as many nodes as with the uniform model.

It is explanable why as k increases the gap between PURE and SMART is narrower. This is because of the condition $u_{\mathcal{C}}(q') \supseteq u_{\mathcal{C}}(q)$ when deciding whether to replicate a new query q at further nodes (see Subsection IV-B). With higher k, the number of such queries q decreases, thus less saving in query replication.

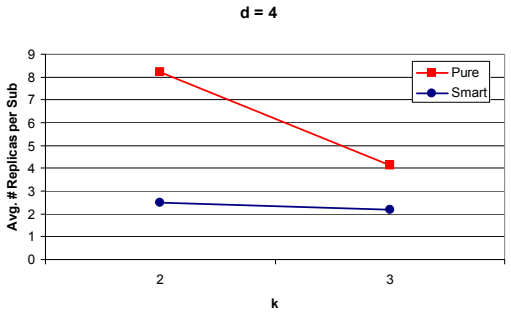
Figure 5 summarizes the skewness of subscription-load distribution by its standard deviation, where different results are observed between the uniform query model and the Pareto query model. While SMART distributes the load across the network significantly better than PURE in the uniform model (Figure 5(a)), the result is mixed for the Pareto query model (Figure 5(b)). In the latter model, there is no difference between PURE and SMART when $d = 4$. When $d = 8$, PURE distributes the load better than SMART when $k \leq 3$. However, as k increases, so is SMART's distribution fairness and when k reaches 3 or above, SMART distributes better than PURE, albeit not as significantly as in the uniform model.

B. Event Notification Efficiency

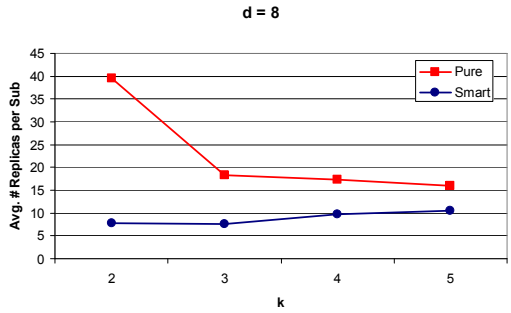
We call it an event when a new data object is published. In the PURE strategy, each event is sent to only one node and guaranteed to find all of its matching subscriptions. In SMART, because we try to avoid subscription replication, each



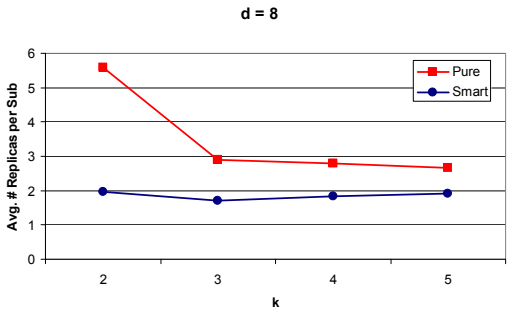
(a) Uniform Query Model: $d = 4$



(b) Pareto Query Model: $d = 4$

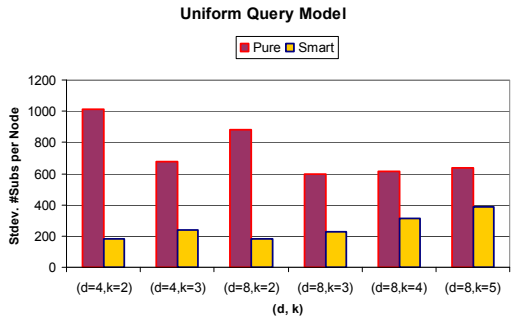


(c) Uniform Query Model: $d = 8$

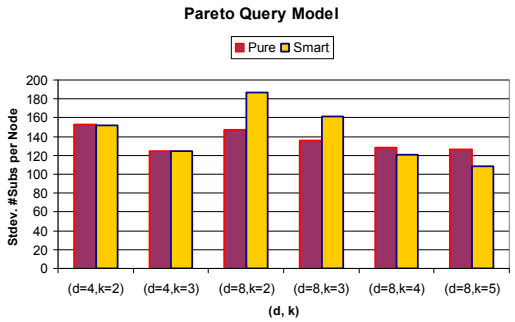


(d) Pareto Query Model: $d = 8$

Fig. 4. Number of replicas per subscription



(a) Uniform Query Model



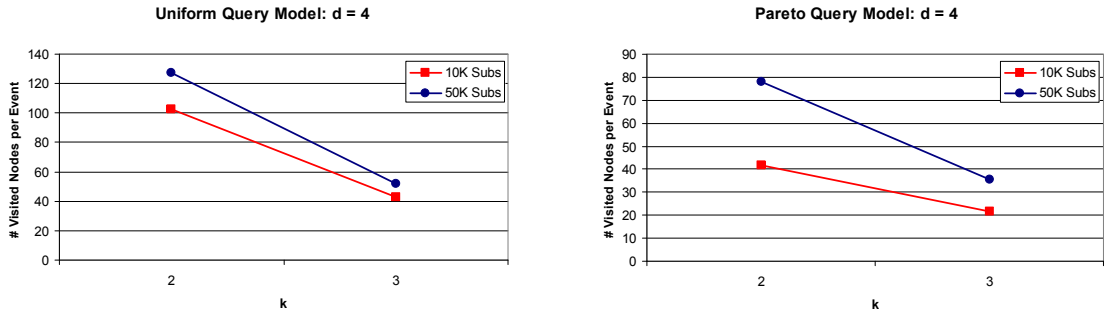
(b) Pareto Query Model

Fig. 5. Standard Deviation of Subscription Load per Node

event should be sent to more nodes to find its matching subscriptions; hence, some extra communication cost. To estimate this cost, we generate 10,000 events, uniformly distributed in the unit cube of d dimensions. We consider both subscription sizes, 10,000 and 50,000. Figure 6 plots the average number of nodes hit by each event.

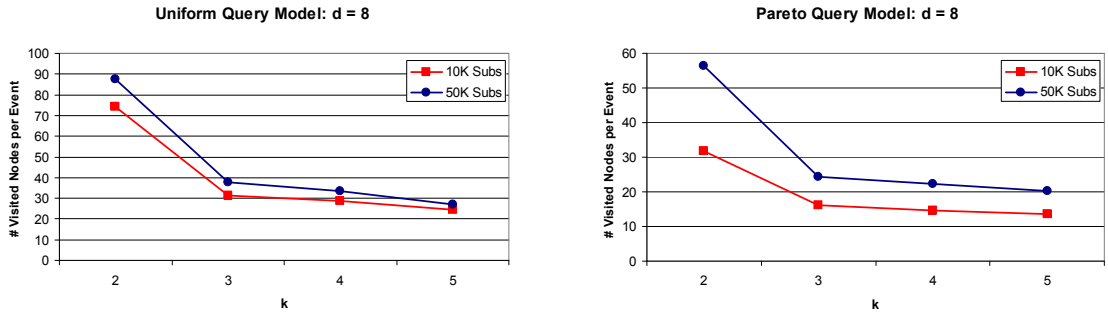
It is observed that there is a significant reduction when k increases from 2 to 3, and then little changes as k gets higher. When $k = 2$, the number of nodes visited by an event on average is less than 130 nodes (or 13% of network size) in the uniform case, and 80 nodes (8%) in the Pareto case. For these cases, respectively, When k is above 2, these numbers are less than 60 (6% of network size) and 40 (4%). This study illustrates that although more than 1 node is visited to find all subscriptions matching an event, on average, the number of such nodes is a small portion of the network size (less than 13% in all cases). For applications with more subscriptions than events, SMART is an efficient solution.

The fact that an increase in k mostly results in a better notification efficiency can also be explained. In CAN, a higher k implies a higher number of neighbors per node, thus usually more subscriptions per node. This implies that during the



(a) Uniform Query Model: $d = 4$

(b) Pareto Query Model: $d = 4$



(c) Uniform Query Model: $d = 8$

(d) Pareto Query Model: $d = 8$

Fig. 6. SMART: Number of nodes visited in the notification upon an event

notification process, we should visit a smaller number of nodes to find all the matching subscriptions.

C. Increasing the Number of Subscriptions

When more subscriptions are submitted to the network, the replication cost remains unchanged with PURE, as illustrated in Figure 7(a) (the uniform query model) and Figure 7(b) (the Pareto query model). However, the replication cost actually is improved if we use SMART (see Figures 7(c) and 7(d)). This is because the more subscriptions submitted, the more likely projection-covering relationships, and, consequently, more replication savings.

In terms of the notification efficiency, considered in SMART, a five-fold increase in the number of subscriptions results in only slightly more visited nodes per event (Figure 6), especially for the uniform query model. For the Pareto model, increasing the number of subscriptions has more of an impact on the number of nodes hit per event. However, except for $k = 2$ where the number of nodes is roughly doubled, the number of additional nodes hit is no more than 50%.

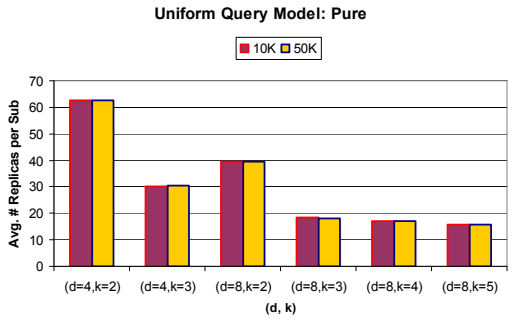
The results of this study show that SMART's replication cost and notification efficiency are sustainable under increasing sizes of subscriptions, which is a desirable property of any large-scale pub/sub system.

VI. Conclusions

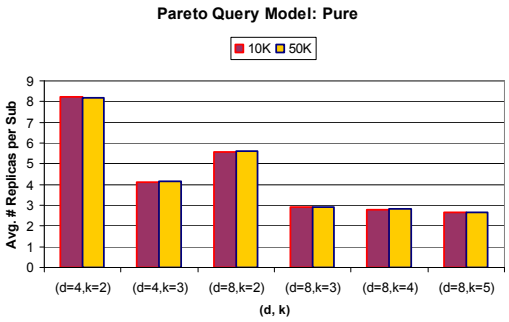
We have presented a random projection approach to deploying pub/sub services in CAN-based P2P networks. Hashing subscriptions from a high dimension to a low dimension so they can be stored and matched with data objects efficiently in the CAN network is not trivial. Not only need we preserve data locality, but also subscription locality. In addition, a desirable property of any distributed pub/sub system is to minimize the replication of subscriptions.

Random projection is a well-known method but our work is the first to explore its applicability in a CAN-based pub/sub network. We have investigated two strategies, PURE and SMART. PURE simply projects a subscription query onto a random space with the same dimension with CAN and stores the query in the CAN zones intersecting this projection. SMART reduces the subscription load in the network by avoid replicating subscriptions whose projection into the CAN space is covered by another subscription's projection. Furthermore, although SMART incurs a non-optimal cost for the notification process, this cost is modest and sustainable with the subscription size increases. We would recommend SMART for applications where the subscriptions outnumber the publications of new data objects.

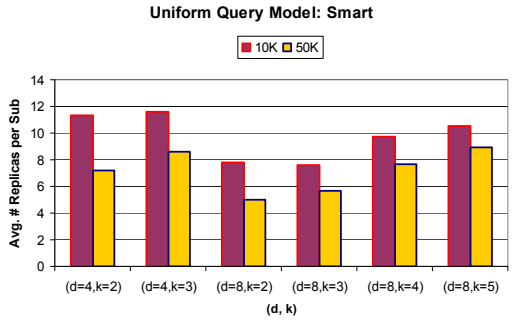
Our future work includes more investigation on the subscription load balancing issue and more on the effect of different choices of the CAN dimension and data dimension.



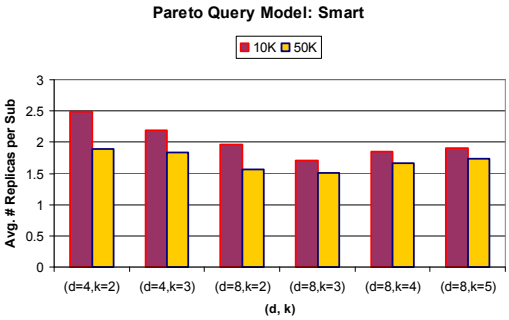
(a) Uniform Query Model: PURE



(b) Pareto Query Mode: PURE



(c) Uniform Query Model: SMART



(d) Pareto Query Model: SMART

Fig. 7. Effect of the number of subscriptions: 10K subscriptions vs. 50K subscriptions

ACKNOWLEDGMENT

The authors acknowledge the NSF for its support of this work under Grants CNS-0615055 and CNS-0753066.

REFERENCES

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *ACM SIGCOMM*, San Diego, CA, August 2001, pp. 161–172.

[2] I. Stoica, R. Morris, D. Karger, M. Kaashock, and H. Balakrishman, "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *ACM SIGCOMM*, San Diego, CA, August 2001, pp. 149–160.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.

[4] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, , and J. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, January 2004.

[5] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1998, pp. 604–613.

[6] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-peer information retrieval using self-organizing semantic overlay networks," in *ACM SIGCOMM*, Karlsruhe, Germany, 2003.

[7] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," in *Conference in modern analysis and probability*. Amer. Math. Soc., 1984, pp. 189–206.

[8] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, vol. 20, no. 8, pp. 1489–1499, 2002.

[9] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot: content-based publish/subscribe over p2p networks," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 254–273.

[10] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann, "A peer-to-peer approach to content-based publish/subscribe," in *DEBS '03: Proceedings of the 2nd international workshop on Distributed event-based systems*. New York, NY, USA: ACM Press, 2003, pp. 1–8.

[11] I. Aekaterinidis and P. Triantafillou, "Internet scale string attribute publish/subscribe data networks," in *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM Press, 2005, pp. 44–51.

[12] S. Voulgaris, E. Rivire, A.-M. Kermarrec, and M. van Steen, "Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks," in *5th Int'l Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[13] S. Bianchi, P. Felber, and M. Gradinariu, "Content-based publish/subscribe using distributed r-trees," in *Euro-Par*, 2007, pp. 537–548.

[14] D. Achlioptas, "Database-friendly random projections," in *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM, 2001, pp. 274–281.

[15] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: applications to image and text data," in *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 245–250.

[16] S. Dasgupta, "Experiments with random projection," in *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 143–151.