Mirinae: A Peer-to-Peer Overlay Network for Large-Scale Content-Based Publish/Subscribe Systems

Yongjin Choi and Daeyeon Park
Department of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology (KAIST)
yjchoi@sslab.kaist.ac.kr, daeyeon@ee.kaist.ac.kr

ABSTRACT

Content-based publish/subscribe systems provide a useful alternative to traditional address-based communication due to their ability to decouple communication between participants. It has remained a challenge to design a scalable overlay supporting the complexity of content-based networks, while satisfying the desirable properties large distributed systems should have. This paper presents the design of *Mirinae*, a new structured peer-to-peer overlay mesh based on the interests of peers. Given an event, Mirinae provides a flexible and efficient dissemination tree minimizing the participation of non-matching nodes. We also present a novel ID space transformation mechanism for balancing routing load of peers even with highly skewed data, which is typical of the real world. Mirinae can be used as a substrate for content-search and range query in other important distributed applications.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; E.1 [**Data Structures**]: Distributed data structures

General Terms

Algorithms Design

Keywords

Peer-to-peer systems, publish/subscribe, overlay multicast

1. INTRODUCTION

Publish/subscribe has become increasingly popular in building large-scale distributed systems. While traditional synchronous request/replay communication needs an explicit IP address of the destination, publish/subscribe systems deliver a message to all, and only those, interested clients based on its content and their subscription set. Thus, publishers do not need to be aware of the set of receivers that vary according to a given message. Example applications that can benefit from this loosely coupled nature of publish/subscribe are news distribution, e-Business (e.g. auction), multiplayer online games, system monitoring, and location-based service for mobile devices.

To use publish/subscribe communication service, receivers register subscriptions that represent or summarize their interests. They will be notified of an event that matches their interests. Based on the selective power of the subscription language, publish/subscribe can be classified [1] as :

- Topic-based. Topics or subjects are used to bundle peers with methods to classify event content. Participants publish events and subscribe to individual topics selected from a predefined set.
- Content-based. A subscription can specify any predicate (or filter) over the entire content. In distributed content-based systems, the subscription is flooded to every possible publishers. A published event is forwarded if a neighboring node has a matching predicate. Thus, content-routing is the process of finding a multicast tree for event dissemination.

Although a topic-based system can be implemented very efficiently, many of modern applications require a content-based publish/subscribe with high expressiveness. However, as mentioned in [1], scalability and expressiveness are conflicting goals that must be traded-off.

Most content-based systems employ an overlay network of event brokers, which support rich subscription languages (e.g. SIENA [1], Gryphon [2]). However, they commonly have the two drawbacks as follows. First, state of the art systems have static overlay networks consisted of reliable brokers under the administrative control, or assume that a spanning tree of entire brokers is known beforehand. Clearly, this is not feasible when a system involves an enormous number of brokers, which join and leave the overlay network dynamically. In extreme case applications, publish/subscribe systems may be organized only by client nodes without any specialized brokers. Second, a broker keeps a large amount of routing states and its control message overhead is huge. This is because every broker can be an intermediate router on the paths of an event dissemination tree. Although the covering relation between subscriptions can reduce this overhead by aggregating them, an unsubscription may have to forward more specific subscriptions covered by it. Hence, the total control traffic effectively has a flooding overhead.

Recently, content-based systems over peer-to-peer (P2P) networks are proposed [3, 4, 5] to solve these problems. P2P networks (e.g. Chord [6], CAN [7], Pastry [8]) address the desirable properties that distributed systems should satisfy. They employ the event broker whose nodeId is the hash of an event type (or a topic name) as the *rendezvous node* (RN), or use P2P routing substrate for the event dissemination tree. The former only provides limited expressiveness obviously. And the latter generates overly large subscription states, since every broker should maintain the predicates of the subscribers whose P2P multicast paths traverse it. In other words, a node may have to participate in routing the events even though it has no interest in them. Also, current approaches lack two essential mechanisms for publish/subscribe over P2P networks, efficient tree construction and load balancing. In conclusion, a distributed publish/subscribe needs a structured overlay network, but it must be designed with great care since the underlying peer-to-peer overlay substrate has a significant effect on the performance of the system.

To solve the above challenges, we propose a new peer-to-peer overlay, called Mirinae, suitable for large-scale publish/subscribes. The design guidelines are (i) a mesh like structure rather than a tree is preferred for building more efficient dissemination tree according to a given event, (ii) a node neighbors with the nodes whose interests are similar to its interest in the overlay network in order to minimize the participation of non-matching nodes in the dissemination. Starting from these guidelines, Mirinae organizes the overlay network based on the predicate summary ID of each node rather than the complex selection predicate itself. The core contributions of this paper include:

- A new peer-to-peer algorithm that construct a mesh based on the similarity of interest of nodes.
- An algorithm to build a proximity aware event dissemination tree whose maximum delivery depth and fan-out of a forwarding node are limited to $O(\log N)$.
- A novel ID space transformation for balancing load and reducing false positives in realistic workloads.

The rest of this paper is structured as follows. In what follows, we describe our basic design and the operations of Mirinae in Section 2. Section 3 presents ID space transformation algorithm that improves performance of the basic protocol. In Section 4, we present some preliminary simulation results. Finally, Section 5 concludes.

2. DESIGN

Mirinae is a scalable overlaynetwork for large-scale publish/subscribes. Mirinae design centers around a virtual *hypercube*, which is a generalization of a three-dimensional cube into d dimensions. Each node in the Mirinae has a d-bit identifer based on its subscription predicate and is mapped onto a vertex in the logical hypercube. An event is transformed to an ID space specifying the set of IDs whose predicates satisfy the event. Therefore, the event dissemination problem is transformed to identifying the multicast tree spanning the given event ID space. The spanning tree is one of embedded trees in the hypercube-like mesh of Mirinae. One important advantage of our approach is that the participation of non-matching nodes can be minimized. This section describes the basic pieces of our design beginning with the description of the data model.

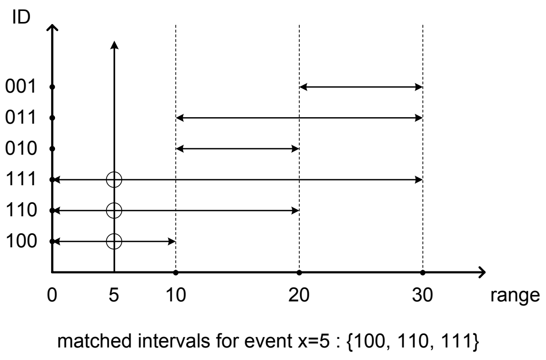

Figure 1: An example of the ID function.

2.1 Data Model

For content-based publish/subscribes, Mirinae needs an ID function to model the predicate of nodes and events. ID function maps a predicate to an *ID* (point) and maps an event to an *ID Space* (range of IDs). Mirinae disseminates an event using the eID (event ID Space). For uniqueness of IDs, the ID of a node is a concatenation of predicate summary ID and a node uniquifier (i.e. IP address).

We define the relation between ID, eID and ID function f_{ID} as follows.

$$\begin{aligned} Predicate_{N_i} &= \bigcup_{k}^{interested} \{event_k\} \\ ID_{N_i} &= f_{ID}(Predicate_{N_i}) \\ eID_{event_i} &= f_{ID}(event_i) \end{aligned}$$

A node has the ID that *covers* the predicate of the node at a minimum since the unique requirement of ID functions is that false negatives cannot be allowed. The ID function should have the following property;

$$\begin{aligned} \forall i, j, \quad if \quad & event_j \in Predicate_{N_i},\ then \\ & eID_{event_j} \ni ID_{N_i} \end{aligned} \tag{1}$$

It means that eID of an event should be the set of IDs of matching predicates.

Figure 1 presents a simple 3-bit ID function for one attribute $0 \le x < 30$. Let the ID function be that Step 1) divide the range of the attribute values by the number of bits and name each, Step 2) set individual bit to 1 if the predicate contains some value of the range that the bit represents (*covers*). By this ID function, a predicate, $2 \le x < 15$, has $ID_{N_i} = 110$ [1] and an event, $x = 5$, has $eID = 1{*}{*} (= 100, 110, 111)$ [2].

While an ID function satisfying Eq. 1 can be used in Mirinae, all possible ID functions do not promise the same performance. A good ID function generates IDs whose distri-

[1]Figure 1 only shows the predicates of the form $a \le x < b$. In general, a predicate is expressed as a disjunction of conjunctions of elementary constraints as defined in [1]. It is trivial to get IDs of such predicates in the example.

[2]It is not necessary that an eID is expressed as a single ID space vector.

bution is uniformly random. Unfortunately, the interest of nodes are highly skewed [9] in the real world. So, the ID space is also likely to be skewed and this results in inefficient routing and load imbalance unless the ID function is designed carefully with statistical information about users' interests. Fortunately, Mirinae has a novel mechanism to remedy this as will be described in Section 3.

Efficient indexing for range search has been a hot area of the database system research. High dimensional indexing [10] and efficient interval indexing [11] algorithms will be helpful to Mirinae application programmer to make a good ID function. Bloom-filter based approaches [12, 13] and subscription partitioning methodology [9] can be alternatives. However, the more technical issues for indexing are beyond the scope of this paper.

2.2 Event Dissemination

To maintain the virtual hypercube topology, a Mirinae node has an *ID Cover* that indicates the volume of IDs the node is responsible for. In Mirinae, two nodes are neighbors if their *ID Cover*s have 1 hamming distance. The hamming distance of two ID Covers is defined as the number of different bits except wildcards ('$*$'). Each entry of the routing table called *ID Cover Table* has the physical address, (ID, ID Cover) and network "distance" of a neighbor, which is used for constructing an efficient event dissemination tree. We explain the construction and maintenance of the table in following sections.

As mentioned earlier, an event dissemination is the process of finding the spanning tree that traverses all the nodes whose ID is within the given eID. This consists of the two phases: the first phase is *routing* to find some node whose ID matches the eID and the second phase is *multicasting* from that node with a proximity-aware binomial tree.

Algorithm 2.1: ROUTE(eID)

```
// $n$: current node
// $n$.foo(): the function foo() is invoked at $n$

if ((d = dist(idc, eID)) $\neq$ 0){                                    (i)
  // find the node whose hamming distance is shortest
  for each (nbr in $n$.IDCoverTable){
    if ((d = dist(nbr.IDCover, eID)) < min){
      min = d;  nexthop = nbr;
  }}
  nexthop.ROUTE(eID);
}
```

Algorithm 2.2: MULTICAST(eID)

```
if (eID $\supseteq$ n.ID)
  receive the event if it satisfies the predicate of $n$;

// pick the neighbors whose ID Covers match eID
for each (nbr in $n$.IDCoverTable){
  if (dist(nbr.IDCover, eID) = 0)
    insert nbr into clist;  //an ordered list w.r.t the cost metric
}

// The following is conceptional, will be substituted with 2.3
for each (nbr in clist){
  eID$'$ = split($n$.ID, nbr.ID, eID);                                (ii)
  nbr.MULTICAST(eID$'$);
}
```

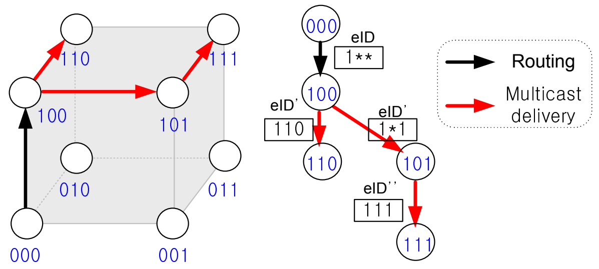

Figure 2: Event dissemination process.

The routing procedure is shown in pseudo code form in Algorithm 2.1. **dist**(A, B) at (i) returns hamming distance between A and B. An event is forwarded to the node whose ID Cover is closest in hamming distance among neighbors until finding a matching node.

Once reached a matching node, the event is multicast with the tree constructed as shown in Algorithm 2.2. **split**(A, B, eID) at (ii) divides the ID space of eID at the first different bit between A and B, and returns each of halved ID spaces to A and B. A node that receives the event forwards it only to the neighbors whose ID Covers match with eID. At the same time, the node splits eID space and assign each of the results to the neighbors. The neighbors are responsible for the delivery of the event to the nodes in their split eID. The event is delivered recursively until all interested nodes receive it. A node engaged in the split process earlier gets more portion of eID since **split()** divides the ID space binomially. In Mirinae, a node with better metric is given a higher priority in the split process among matching neighbors. Therefore the resulting tree is efficient with respect to the metric in the ID Cover table.

Figure 2 shows an example of event dissemination in the 3-dimensional Mirinae. Assume that an event occurs at node 000 and the eID is 1$**$. The event is routed to the first matching node 100 and then multicast from it. The ID spaces on arrows show that eID is split as the event moves toward the leaves of the multicast tree. In this example, node 101 is assigned the responsibility for delivering the event to 111 since node 101 has a better metric than 110. It is noteworthy that no node not interested in the event is involved in the multicast of it, which means that the event is disseminated in a near-perfect manner.

Unfortunately, 'split'-based multicast algorithm may incur duplicated notifications since each node is responsible for range of IDs, namely, its IDCover. To avoid duplication, every intermediate node subtracts all ID Covers of it and its neighbors from eID before multicasting as in Algorithm 2.3. For general addition and subtraction of ID spaces, Mirinae has a special data structure called Binary ID Space Tree. Due to space limitation, we omit the detailed algorithms.

Algorithm 2.3: ASSIGN(eID)

```
eID = eID − {ID Covers of $n \cup n$.nbrs};
while (eID $\neq \emptyset$){
  $nbr$ ← pop a front nbr from clist;
  eID$'$ = {$e$ | $e \in$ eID, dist($e$, $nbr$.IDCover) = 1};
  eID = eID − eID$'$;
  if (eID$'$ $\neq$ null || eID $\supseteq$ $nbr$.ID)
    $nbr$.DISSEMINATE(eID$'$);
}
```

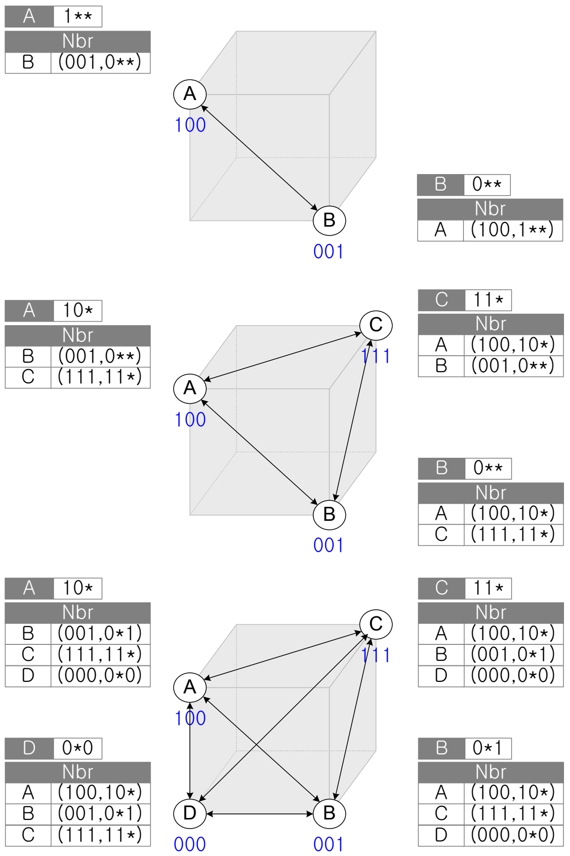

Figure 3: An example of subscribing.

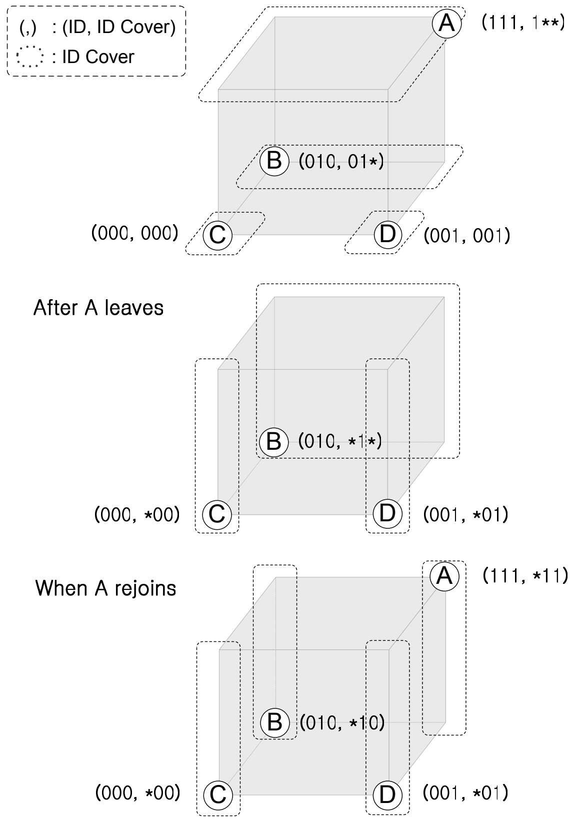

Figure 4: An example of unsubscribing and ID space transformation.

Due to the properties of binomial trees, the expected number of event notification steps and the maximum fan-out, which the first matching node has, are $O(\log N)$, where N is the number of matching nodes in the Mirinae network.

2.3 Subscribing (Join)

When a new subscriber joins the publish/subscribe network, it contacts a node already in the network with its subscription ID. Next, using routing mechanism, the new node must find the node whose ID Cover covers the ID. The current occupant node splits its ID Cover in half and assigns one half to the new node. The new node learns its hamming neighbor set from the previous occupant since the set is a subset of the previous occupant's neighbors, plus that occupant itself. Finally, both the new and old nodes' neighbors must be informed of the change of ID Covers.

For the comprehension, Figure 3 shows a step-by-step example, where the dimension is 3. While not explicitly proven here, a joining or leaving node causes $O(\log N)$ messages to keep Mirinae protocol invariants as other P2P protocols.

2.4 Unsubscribing (Leave) and Node Failure

Unsubscribing procedure is the exact opposite to the subscribing so that a neighbor inherits the ID Cover of the leaving nodes. If the ID Cover of the leaving node cannot be merged with the ID Cover of one of its neighbors, the departing node's ID Cover should be split and be handed over in order that each neighbor has a valid single ID Cover to prevent the ID space of nodes from fragmented. Fortunately, a leaving node is guaranteed to know the neighbors that must take over its ID Cover fragments. Figure 4 shows an example of this.

A node failure is detected by periodic heartbeat messages between neighbors. The message contains the ID Cover of a node, a list of its neighbors and their ID Covers. The first node that perceives a node failure immediately takes over the ID Cover of the failed node and performs the same procedure of unsubscribing.

3. EFFICIENCY IN THE REAL WORLD

The basic Mirinae algorithms as described in Section 2 can provide efficient (logarithmic) performance only with a well-designed ID function or uniformly distributed subscriptions. However, those conditions can be easily violated in many real world distributed applications. To address this difficulty, we propose an ID space transformation algorithm to tackle non-uniformity in two aspects.

3.1 Load Balancing

Figure 4 shows an example of skewed ID space partition. Node A has the four times bigger ID Cover than C and D. This problem is significant since real workloads have Zipf-like distribution rather than uniform [9] and the critical design choice we made is to eliminate cryptographic hashing (i.e. SHA-1) in assigning IDs. In Mirinae, the ID Cover size of a node is proportional to forwarding load in routing and dissemination and the number of neighbors. To resolve this, Mirinae uses a *leave-join* style ID space transformation. The basic idea of transformation is shown in Figure 4. When node A leaves and rejoins the network, all 4 nodes have the ID space of the same size and 2 neighbors eventually. Similar leave-join based load balancing mechanisms are proposed recently [14, 15, 16]. They estimate a load distribution of the system and move some lightly loaded nodes to heavily loaded region of the space by changing IDs of nodes. Mirinae, on the other hand, moves the ID Cover fragments of a heavily loaded node without any change of IDs.

Every node can decide easily whether the ID space is skewed or not by checking the number of its neighbors that must take over its ID Cover fragments when it leaves. When the ID space is uniformly partitioned, almost every node has one sibling node whose ID Cover can be merged with the departing node's ID Cover. So, having several neighbors that perform the takeover means that ID space is skewed. After deciding whether a transformation is needed, each node emulates leave-join process and its ID Cover fragments are handed over to some neighbors. The neighbor receiving an ID fragment now has a larger ID Cover. By repeating this transformation independently at the nodes with larger ID Covers, the entire ID space can be near-uniform.

Unfortunately, the ID space transformation for balancing ID Covers is not always possible. If the ID of node A is 100 (and ID Cover $1{*}{*}$) in the example of Figure 4, the ID space cannot be made perfectly uniform under the constraint that the ID Cover of each node is a single vector. However, the entire ID space is sufficiently extensive compared to the number of nodes [3]. Hence, Mirinae can find the dimensions in which the skewed ID spaces can be transformed to be uniform in most cases.

3.2 Reducing False Positives

One important metric in evaluating distributed content-based systems is the percentage of *false positives* over the total number of messages. A false positive is defined as a message received by the node not interested in the message.

The main reason why a false positive occurs in Mirinae is shown in Figure 5. When the eID space is partially overlapped with the ID Covers of intermediate nodes, there is some possibility of false positives since the IDs of those nodes may not be contained within the eID space. To suppress those false positives, the ID Covers of intermediate nodes in the dissemination path should have one of the two relations with the eID, completely containment or disjoint. Based on this observation, Mirinae utilizes a dimension transformation mechanism to reduce false positives for popular events. The basic idea is very similar to the mechanism in Section 3.1. By merge-split process, Mirinae aligns ID Covers along the dimensions in which the eIDs of popular events are specified. In Figure 5, a false positive occurs for an event $*0*$ at node B (or C according to the proximity) in the initial stage. When a node detects that the events of $*1*$ or $*0*$ occur frequently, it starts a dimension transformation process with its neighbors. The ID Covers of B and D are now merged and re-split along the second dimension. A and C also do the same process. As a result, there is no false positive in dissemination of the events whose eIDs have non-wildcards in the second dimension.

[3]We used 160 bit identifier composed of 128 bit predicate summary and 32 bit uniquifier. The size of dimensions (d) can be chosen large enough since the performance of peer-to-peer systems including Mirinae is independent of d.

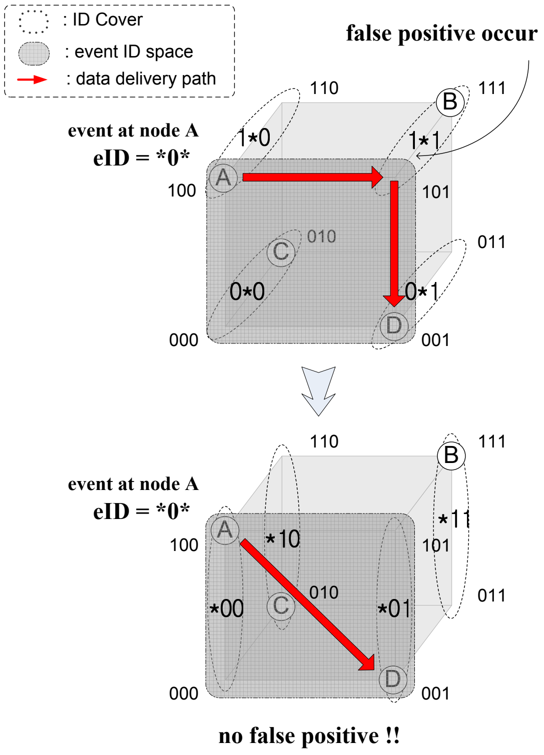

Figure 5: An example of reducing false positives by dimension transformation.

4. PRELIMINARY EVALUATION

This section presents some preliminary evaluation of the Mirinae protocol using simulations. We implemented a discrete event-based simulator, which does not model any delays or bandwidth on links for simplicity.

We omit the results on basic performance metrics such as path length and the number of neighbors, and the maximum delivery depth and fan-out of dissemination trees, since it is easy to prove that they are $O(\log N)$ with high probability.

Our evaluation is focused on two features of the Mirinae system: efficient dissemination of events, and balancing of routing load indicated by the size of ID Covers. We assume without loss of generality that an application domain schema has a single **float** attribute with range $[0, 1]$. We use a simple ID function that uniformly partitions the range of values. The number of nodes in the network, N, is 4096 and the size of dimensions, d, is 160. However, the results in this section has little relevance to those parameters.

Figure 6 shows one of the most important performance metric, false positives, for about 25,000 events. In general, the false positive is a function of event popularity (i.e. the ratio of interested nodes over all nodes). The basic Mirinae protocol incurs a lot of false positives for highly selective events. This is because the eID indicating a small region of ID space has high probability that the traversed nodes during dissemination have the IDs not contained within the eID although their ID Covers match the eID. Those false positives are fairly reduced by the split dimension transformation. As a result, Mirinae can minimize the participation of non-interested subscribers irrespectively of the event popularity.

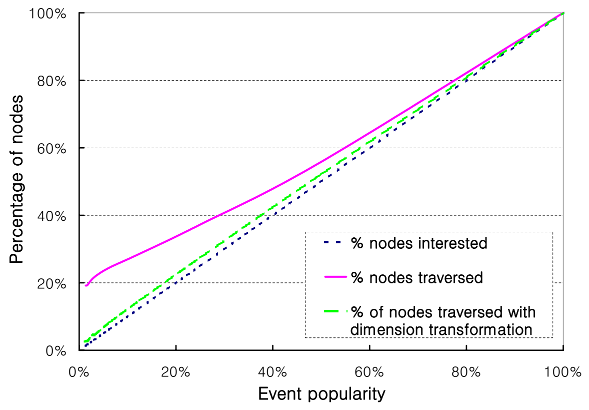

Figure 6: Percentage of nodes traversed.

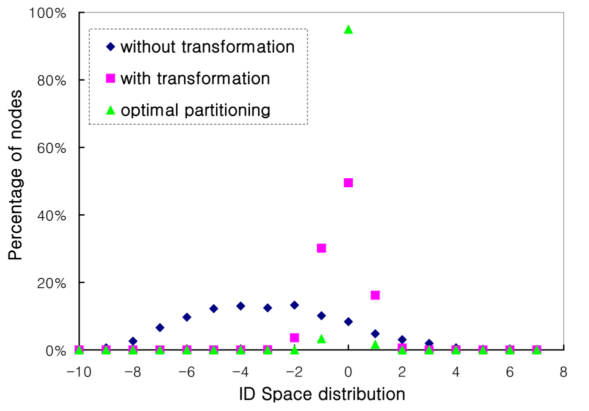

Figure 7: Effect of ID space transformation with Zipf-like subscriptions ($\alpha = 1.0$).

Obviously, the best way to balance routing load is to design an ID function whose resulting IDs are uniformly distributed. However, Mirinae can balance load even though ID functions do not have such property. The effect of the Mirinae load balancing mechanism are shown in Figure 7. The average size of ID Covers is denoted by 0 on the x-axis. With Zipf-like subscriptions, ID Covers are highly-skewed. However, the ID space transformation makes the distribution of ID Covers near-uniform. This ensures that Mirinae preserves efficiency and scalability in realistic workloads.

5. CONCLUSION

In this paper, we have presented a new peer-to-peer overlay called Mirinae suitable for large-scale publish/subscribe systems. Mirinae can construct a flexible and efficient event dissemination tree with minimum participation of the nodes that have no interest in the event. At the same time, it has the desirable properties such as self-organization, scalability, and bounded delivery depth. Finally, we propose novel a ID space mechanism for balancing load and reducing false positives in realistic workloads.

We believe that Mirinae can be used as a underlying substrate in other important distributed applications such as collaborative data distribution, interest-based streaming service, and multi-attribute range queries.

6. ACKNOWLEDGMENTS

We would like to thank Yongjoo Song, Donggook Kim, Hyunbin Lee, and Keuntae Park for their valuable suggestions and insightful comments. We also want to thank the anonymous reviewers for their helpful feedback.

7. REFERENCES

[1] A. Carzaniga and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. *The 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, pages 262–272, May 1999.

[3] M. Castro, P. Druschel, A. M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized publish-subscribe infrastructure. In *Proceedings of the 3rd International Workshop on Networked Group Communication (NGC'01)*, volume 2233, pages 30–43, Nov. 2001.

[4] P. R. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. *The 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, Jun. 2003.

[5] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. *The 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, Jun. 2003.

[6] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, San Diego, California, USA, 2001.

[7] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 161–172, San Diego, California, USA, 2001.

[8] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Hiedelberg, Germany, Nov. 2001.

[9] Y.-M. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H. J. Wang. Subscription partitioning and routing in content-based publish/subscribe systems. In *16th International Symposium on DIStributed Computing*, 2002.

[10] J. Goldstein, J. C. Platt, and C. J. C. Burges. Indexing high dimensional rectangles for fast multimedia identification. Technical report, Microsoft Research, 2003.

[11] K.-L. Wu, S.-K. Chen, P. S. Yu, and M. Mei. Efficient interval indexing for content-based subscription e-commerce and e-service. In *IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, Aug. 2004.

[12] P. Triantafillou and A. Economides. Subscription summaries for scalability and efficiency in publish/subscribe systems. *The 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, Jun. 2003.

[13] G. Koloniari and E. Pitoura. Bloom-based filters for hierarchical data. In *the 5th Workshop on Distributed Data and Structures (WDAS)*, Jun. 2003.

[14] P. Ganeshan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *Very Large Databased (VLDB)*, 2004.

[15] D. Karger and M. Ruhl. Simple efficient load-balancing algorithms for peer-to-peer systems. In *Third International Workshop on Peer-to-Peer Systems*, 2004.

[16] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of the 2001 ACM SIGCOMM Conference*, 2004.