HOMED: A Peer-to-Peer Overlay Architecture for Large-Scale Content-based Publish/Subscribe Systems

Yongjin Choi, Keuntae Park and Daeyeon Park
Department of Electrical Engineering & Computer Science
Korea Advanced Institute of Science and Technology(KAIST)
Yusung-Gu, Taejon, Korea
{yjchoi, ktpark}@sslab.kaist.ac.kr, daeyeon@ee.kaist.ac.kr

Abstract

Content-based publish/subscrbe systems provide an 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 a new peer-to-peer overlay called HOMED for distributed publish/subscribe systems. It can construct a flexible and efficient event dissemination tree by organizing participants based on their interest. The delivery depth of an event as well as subscribing/unsubscribing overhead scales logarithmically with the number of participating nodes.

1. Introduction

Publish/subscribe has become increasingly popular in building a 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 varies according to a given message. Example applications that can benefit from this loosely coupled nature of publish/subscribe are news distribution, e-Business (eg. auction), multiplayer online games, system monitoring, and location-based service for mobile devices.

To use a publish/subscribe communication service, receivers register subscriptions that represent or summarize their interests. In turn, 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 [3] 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 of the publication. 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 an event dissemination tree.

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 [3], 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 (eg. SIENA [3], Gryphon [1]). 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 state 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 based on peer-to-peer (P2P) networks are proposed [5] [12] [9] [2] to solve these problems. P2P networks (eg. Chord [11], Pastry [10]) 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. However, the features they exploit from P2P systems are limited to self-organization, fault-tolerance, and guaranteed routing depth. Still, their routing state and control message overhead are enormous, 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. Due to the same reason, the routing depth for notifications is unnecessarily long. 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 architecture has a significant effect on the performance.

In this paper, we propose a new peer-to-peer overlay, called HOMED, suitable for large-scale publish/subscribes. The design guidelines are (i) a mesh like structure rather than a tree is preferred for a reliable and adaptive event dissemination tree, (ii) a node neighbors with the nodes whose interests are similar to its interest in the overlay network in order that only interested nodes participate in disseminating the event. To ease construction and routing, HOMED organizes the overlay network based on the interest digest of each node rather than the complex selection predicate. HOMED can be used not only for flexible topic or type based systems by nature, but also as a routing substrate for highly selective content-based systems. In HOMED, an event is delivered along the path of a binomial tree. Also, the subscribe/unsubscribe overhead is limited to $O(\log N)$.

The rest of this paper is structured as follows. In what follows, we describe our basic design and the operations of a HOMED in Section 2. Section 3 presents some implementation issues and extensions. We summarize our contributions and discuss future plans in Section 4.

2. Design

HOMED (Hypercube Overlay Mesh for Event Dissemination) is a scalable overlay network for large-scale publish/subscribes. HOMED design centers around a virtual *hypercube*, which is a generalization of a three-dimensional cube into d dimensions. Each node in the HOMED has a d bits identifer based on its subscription predicate. Therefore, a node is mapped onto a vertex in the logical hypercube. An event is propagated to interested receivers along an embedded multicast tree in the hypercube. Due to this awareness of interests, HOMED can find the "best" event dissemination tree with minimum overhead.

First, we describe our HOMED in its basic form. In Section 3, we present several extensions that improve the performance and flexibility.

2.1. ID assignment

Every node joins a HOMED with its predicates specifying the events of interest. A HOMED node can be a broker serving many clients or a client itself in brokerless systems. By some ID generating function, a set of predicates (or a predicate) is transformed to a d bit ID (Interest Digest) .The ID is used as the basis of HOMED topology. Although the ID function has some effect on the performance, ID is just a guide-rule to construct an event dissemination tree and support a content-based service.

The unique requirement of the ID function is that its resulting IDs preserve a bitwise covering relation. More precisely, if a predicate α is covered by another predicate β, its digest ID_β must subsume all 1s of ID_α. A simple ID function is that Step 1) split d bits into as many segments as the number of attributes Step 2) divide the range of the attribute values by the number of bits in the segment , and Step 3) set individual bit to 1 if the predicate contains the value the bit represents. For an attribute that specify constrains as a range of values, it is necessary to divide the range into a finite number of intervals and name each.

Bloom-filter based approaches [13] [8] can be interesting alternatives since the bloom-filter also satisfies the above requirement. Subscription partitioning methodology in [14] is also a useful basis of the ID function. In addition, it is often desirable to make an ID based on only a subset of attributes in many applications. However, we will not elaborate on the technical issues here since designing a good ID function depends on the application domain and is beyond the scope of this paper.

2.2. Event Dissemination

To maintain the virtual hypercube topology, every HOMED node has a routing table called *ID cover table* as shown in Figure 1. A HOMED node has an *ID cover* that indicates the set of IDs the node must cover. If all vertices of the hypercube have the corresponding HOMED nodes, the ID cover of each node is the same as its ID. However, as the number of the nodes is much smaller than the number of vertices in the hypercube typically, each node is responsible for larger ID space than its ID.

The ID cover table has d entries, each of them corresponds to an ID whose hamming distance is 1 (i.e. ideal hamming neighbors of the node). Each entry has the physical address and ID cover of the neighbor that covers the corresponding ID. The final column is a scalar metric, such

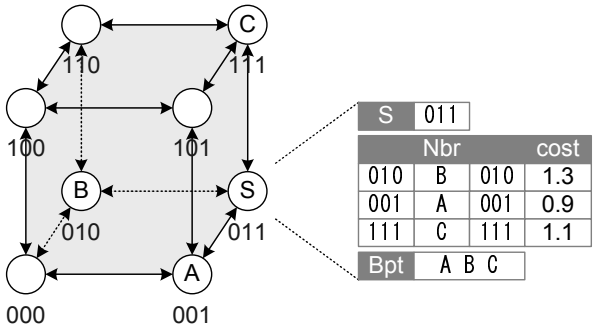

Figure 1. ID Cover Table.

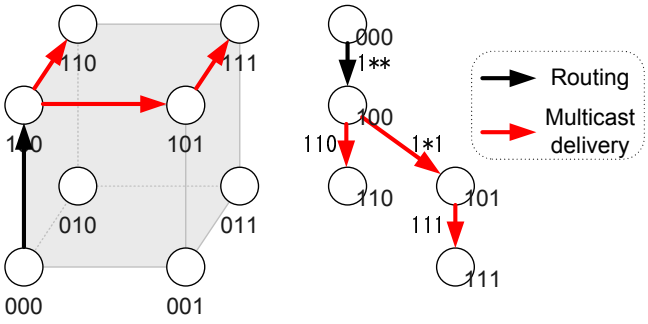

Figure 2. An example of event dissemination.

as the network "distance", which is used for constructing an efficient event dissemination tree. The bottom entry is the list of backpointers that reference the node. When the ID cover of a node changes, the nodes in that list are informed to update or change their neighbors. This will be explained minutely in Section 2.3.

A published event has an eID generated by the ID function. The eID represents the ID space where the event should be delivered. Hence, the dissemination is the process of matching the ID space of the event with nodes' ID. 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 an event dissemination tree.

Algorithm 2.1: ROUTE(eID)

```
ID ← publisher;
while ((d = dist(ID, eID)) ≠ 0){                    (i)
  //find the node whose hamming distance is shortest among neighbors
  for each (node in ID.Nbr){
    if (dist(node, eID) < d){
      d = dist(node, eID);
      ID = node;
  }}
  forward to ID;
}
```

Algorithm 2.2: DISSEMINATION(ID, eID)

```
for each (node in ID.Nbr){
  if (dist(node, eID) = 0)
    insert node into clist;   //an ordered list w.r.t the cost metric
}

if (clist ≠ null){
  for each (node in clist){
    eID′ = split(ID, node, eID);                    (ii)
    DISSEMINATION(node, eID′);
}}
```

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. Wild cards in IDs are ignored in calculating the hamming distance since they mean "don't care". An event is forwarded to the node whose ID is closest in hamming distance among neighbors.

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 resulting split IDspaces to A and B. A node that receives the event forwards it only to the neighbors whose ID is in 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 HOMED, a node with better cost metric is given a higher priority in the split process among interested 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 HOMED. An event occurs at the node 000 and the eID of the event is (1**). The event is routed to the first matching node 100 and then multicast. ID spaces on arrows show that eID is split as the event moves toward the leaves of the multicast tree. In this example, the node 101 is assigned the responsibility for delivering the event to 111 since the node 101 has a better cost metric than 110.

It is notable that no node is engaged in the multicast of the event that it is not interested in, which means that the event is disseminated very efficiently with minimum messages. While not explicitly proven here, suppose that IDs of every nodes are different, the expected number of event notification steps is $O(\log N)$ rather than d, where N is the number of HOMED nodes in the network.

2.3. Subscribing

When a new subscriber enters the publish/subscribe network, it contacts any well-known node with subscription information. From that node, it traces ID cover table to find its position in the network. This participation process is called *subscribing*. During the subscribing, the node gets direct neighbor information and its portion of ID cover so that it can complete its ID cover table.

Algorithm 2.3: SUBSCRIBE(ID)

```
ID.Nbr = ID.Bpt = ∅;  ID.Cvr ={ID};
n ← a well-known node;

//step1: find the position
n ← ROUTE(ID);

//step2: split ID cover and make the ID table of the joining node
ID.Cvr ← split(n, ID, n.Cvr);
L ← {x|dist(x, ID) = 1};   //hamming neighbors (hn)
for each (hn in L){
  for each (node in n.Nbr ∪ n.Bpt){
    if (hn ∈ node.Cvr){
      ID.Nbr ← (hn, node, node.Cvr);
      node.Bpt = node.Bpt ∪ ID;
  }}
  if (hn ∈ ID.Cvr)
    ID.Nbr ← (hn, ID, ID.Cvr);
}

//step3: update neighbor's table
L ← {n} ∪ n.Bpt;
for each (bpn in L){
  for each (hn in bpn.Nbr){
    if (hn ∈ ID.Cvr){
      replace the hn entry with (hn, ID, ID.Cvr);
      move bpn from n.Bpt to ID.Bpt;
    } else if(hn = n)
      hn.Cvr ← n.Cvr;
}}
```

Algorithm 2.3 shows the sequence of subscribing process with pseudo code. It is organized as three steps. At the first step, a subscriber looks for the node that covers the its ID. This is accomplished simply by call **ROUTE** with its ID. The second step is to determine the ID cover of the subscriber. The subscriber gets its ID cover portion by **split**() function. And, the subscriber makes the neighbor list from neighbors and backpointers of the node that previously covered its ID. The split of ID cover incurs the split of backpointer list. As a final step, nodes that point the ID cover which is split from the original node change the neighbor list to point new owner of the ID cover. New and old owners also update their backpointer lists to reflect those nodes.

For the comprehension, we provide a step-by-step example in Figure 3, where the ID size is 3 bits.

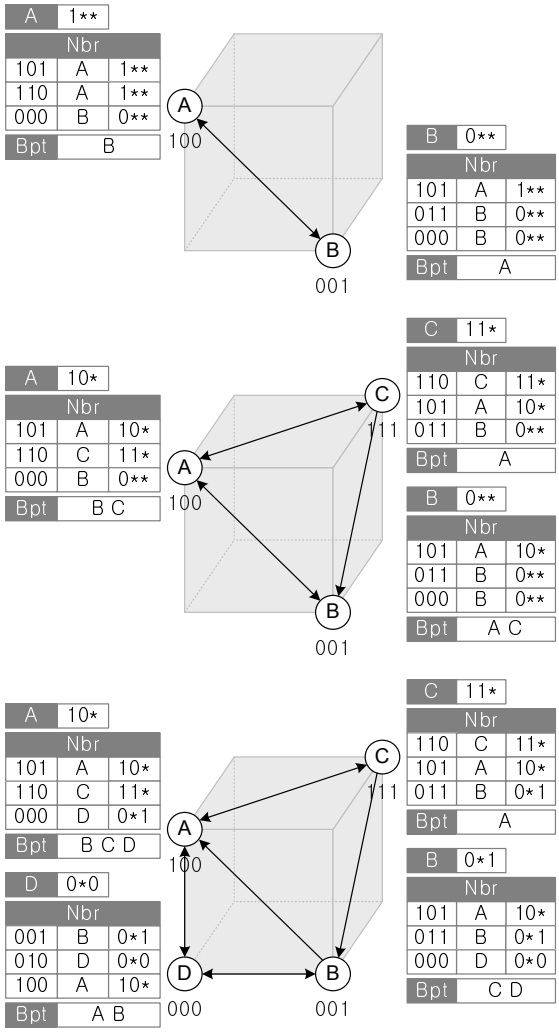

Figure 3. An example of subscribing.

In *subscribing*, number of messages to set up ID cover table is $\log N + 2d$ where N is the total number of subscribers and d is the number of bits in ID. A new subscriber sends just one message to find the node that covers its ID and the message moves $\log N$ hops on average. It also contacts all the neighbor nodes and backpointing nodes of the original owner of the ID cover in order to fill the neighbor list. The number of messages are $2d$ as total number of neighbor nodes and backpointing nodes in the network are the same and the average number per node is d.

2.4. Unsubscribing

Subscribers can leave the subscribe network at anytime, which is called *unsubscribing*. To maintain the HOMED network, other subscribers must cover the ID cover of the leaving node and update their neighbor information. Sub-

scribers that want to leave the network send notification to any one neighbor then it inherits the ID cover of the leaving node.

Each node knows the ID cover table of neighbor nodes that is piggybacked in the periodic heart beat message. The node that succeeds to the ID cover of the leaving node sends notification of ID cover change to all the backpointing nodes of itself and the unsubscribing node. It also informs the neighbors of the leaving node that backpointers to the node are no longer needed. Unsubscribing operation is expressed by pseudo code in Algorithm 2.4.

Algorithm 2.4: UNSUBSCRIBE(leaveID)

```
ID.Cvr = ID.Cvr ∪ leaveID.Cvr;
//step1: redirect references from leaveID to ID
for each (bpn in leaveID.Bpt){
  for each (hn in bpn.Nbr){
    if (hn ∈ ID.Cvr){
      replace the hn entry with (hn, ID, ID.Cvr);
}}}

//step2: eliminate obsolete backpointers
for each (node in leaveID.Nbr)
  node.Bpt = node.Bpt − {leaveID};

//step3: notify ID cover change
for each (node in ID.Bpt){
  for each (nb in node.Nbr){
    if (nb = ID)
      nb.Cvr ← ID.Cvr;
}}
ID.Bpt = ID.Bpt ∪ leaveID.Bpt;
```

During *unsubscribing*, $3d$ messages are delivered by one hop transmission as the node knows where to send ID cover change notifications and obsolete backpointer notifications by hearing neighbor's ID cover table in heart beat message.

2.5. Node Departure

Node departure means a leaving of a node without notification. Reaction mechanism is basically the same as unsubscribing except that neighbors detect the node departure by periodic heart beat messages. The first node that perceives the node departure inherits the ID cover of the leaving node and executes unsubscribing process explained above.

HOMED's mesh like structure gives the benefit of localized recovery of the node departure; only neighbors of the leaving node are involved in the process. While previous approaches need to contact almost all nodes to eliminate obsolete subscription through the reverse path of the tree, HOMED can handle a the node departure with much smaller message and delay overhead.

3. Discussion

In this section, we discuss some design issues and possible improvements.

3.1. Handling duplicate IDs

In HOMED, an ID is not guaranteed to be unique in the network since IDs are determined by interests of nodes. So, several nodes can have the same ID. Although it is desirable to avoid this ID conflict as much as possible with the "good" ID function discriminating interests, HOMED must be able to handle duplicate IDs. One possible solution is that the nodes with the same ID construct an overlay multicast tree [6] [7] rooted at the first joining node or the most powerful one. Another solution is that an ID is composed of interest and identifier of a node. An event also has the corresponding part in its ID, which is normally set to (content ID, ***). In this case, publishers can select receivers among interested ones.

3.2. k-ary Hypercube

In the previous section, an ID is represented by a binary bit vector, and thus the resulting HOMED topology is built on a logical binary hypercube. However, each bit in an ID can be substituted with b bits for the HOMED network that requires a large ID space. [1] The routing table of HOMED is organized with base 2^b in much the same way as that of Pastry [10]. We have the two choices of how many entries a node must have among k ($= 2^b$) possible values.

1. For each row, a node has two neighbors whose ID segment is numerically closest among smaller and bigger ones. The delivery depth is increased to $O\left(2^b \log_{2^b} N\right)$.
2. A node's routing table has $2^b - 1$ entries for each row as in Pastry. While the routing table size becomes $O\left(2^b \log_{2^b} N\right)$, the deliver depth is $O\left(\log_{2^b} N\right)$.

Therefore, there is a trade-off between the size of routing tables and the depth of event delivery trees.

3.3. Supporting Content-Based Systems

HOMED may suffer from the same problem that topic-based systems have because it is based on the simple ID. Let us assume that a node has an ID segment 0110, which represents an integer attribute $0 < x < 100$, but the actual interest of the node is $30 < x < 80$. Then, it will receive

[1] This is similar to that binary hypercubes is generalized to k-ary n-cubes.

unnecessary messages when they are out of the range of its interest. To reduce those false positives, the exact predicate must be known to all other nodes as existing content-based systems do. This will result in unacceptable overhead. However, HOMED can localize the propagation of exact predicates. For the previous example, the node with ID (0110...) sends its predicate only to the nodes that have the same ID segment 0110, since one of them will deliver matching events.

Content-based networking in HOMED has two advantages. First, a predicate is propagated only to a small number of nodes by a subset of subscriber's neighbors, and thus the number of hops in propagation is limited. Second, predicates do not have to be known in time since they are used only for reducing false positives. In HOMED, predicates are piggybacked on heartbeat messages. Based on the aggregated predicates, each node in the event dissemination tree decides matching descendants using algorithm of [4].

4. Conclusion

In this paper, we have presented a new peer-to-peer overlay called HOMED suitable for large-scale publish/subscribe systems. HOMED 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 limited delivery depth and overhead.

Evaluation of publish/subscribe systems is a hard work since they are affected by a lot of components including the probabilistic model of publishers and subscribers. Measuring the performance of HOMED is now in progress.

It is another challenge to exploit physical locality in content-based publish/subscribe. While HOMED already has the scheme to assign different responsibilities to the descendants of a tree, a network-wide optimization is possible using proximity estimation techniques such as IDMaps and GNP. To couple HOMED with them is under study.

References

[1] 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.

[2] A. R. Bharambe, S. Rao, and S. Seshan. Mercury: A scalable publish-subscribe system for internet games. In *Proceedings of the 1st Workshop on Network and System Support for Games (Netgames)*, Braunschweig, Germany, Apr. 2002.

[3] 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.

[4] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of the 2003 ACM SIGCOMM Conference*, Karlsruhe, Germany, Aug 2003.

[5] 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.

[6] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, Jun. 2000.

[7] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, Oct. 2000.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.