MEDYM: Match-Early with Dynamic Multicast for Content-based Publish-Subscribe Networks

Fengyun Cao, Jaswinder Pal Singh

Computer Science Department, Princeton University
Princeton, New Jersey 08540, USA
{fcao, jps}@cs.princeton.edu

Abstract. Design of distributed architectures for content-based publish-subscribe (pub-sub) service networks has been a challenging problem. To best support the highly dynamic and diversified content-based pub-sub communication, we propose a new architectural design called MEDYM - Match-Early with DYnamic Multicast. MEDYM follows the End-to-End distributed system design principle. It decouples a pub-sub service into two functionalities: complex, application-specific matching at network edge, and simple, generic multicast routing in the network. This architecture achieves low computation cost in event matching and high network efficiency and flexibility in event routing. For higher scalability, we describe a novel approach to extend MEDYM to a hierarchy structure called H-MEDYM, which effectively balances the trade-off between event delivery efficiency and server states maintenance. We evaluate MEDYM and H-MEDYM using detailed simulations and real-world experiments, and compare them with major existing design approaches. Results show that MEDYM and H-MEDYM achieve high event delivery efficiency and system scalability, and their advantages are most prominent when user subscriptions are highly selective and diversified.

Keywords - content-based publish-subscribe network, multicast.

1 Introduction

Content-based publish-subscribe (*pub-sub* for short) is an important paradigm for asynchronous communication among entities in a distributed network. In such systems, users *subscribe* to future *events* that are of their interest by specifying complex conditions on event content, and are *notified* when events satisfying the conditions are *published* into the system. For example, a user who subscribes to stock ticker events with condition "*PriceChange* > 10% AND *Volume* > 100m" is notified when a stock has price movement of above 10% or transaction volume of more than 100 million shares. Such timely delivery of customized information is of great value to many distributed applications, and has become an interesting and important research topic.

For scalability and reliability reasons, a large-scale pub-sub system often takes the form of a distributed service network: as shown in Fig. 1, a set of pub-sub servers is distributed over the Internet; clients access the service, either to publish events or to register subscriptions, through servers that are close to them or in the same administrative domains. In this paper, we study the problem of efficient event delivery in the service network, i.e. from servers where the events are published to servers with matching subscriptions. We do not address the "last-mile" event delivery from servers to local clients in this paper.

Efficient event delivery is challenging for two reasons: first, published events do not carry destination address information. Rather, it is the system's responsibility to *match* each event with user subscriptions to identify the servers that are interested in it. Second,

even if the destinations are known, it is not clear how to *route* the events to the destination servers. This is because of the highly diversified user interests in a content-based pub-sub system: every event can match the interest of a different set of servers, and in the worst case, there can be $2^{\#servers}$ such destination sets. How to achieve efficient delivery to so many destination sets is yet an open question.

Existing architecture designs for content-based pub-sub networks typically connect servers into pre-configured overlay networks. Events are routed along the overlay network topology, choosing which connections to follow based on matching results. Because event delivery routes are constrained by the overlay topology, it is inevitable that events are sent to/through servers that are not interested in them, generating extra processing and network load. As analyzed in the paper, such overhead can be especially high when user interests are highly selective and diversified.

In this paper, we explore the possibility of a very different approach. Corresponding to the dynamic communication patterns in pub-sub networks, we propose an architecture called *MEDYM: Match-Early with DYnamic Multicast*. In MEDYM, events are first matched with subscriptions to identify destination servers, and then delivered to destinations along multicast routes computed and constructed on the fly. In this way, MEDYM allows fine-grained optimization for delivery of each individual event. For example, it is able to send events only to the servers that are interested in them, minimizing event traffic load on pub-sub servers. Using configured overlay networks, no existing solution achieves this highly desirable property. MEDYM network is also easy to deploy, and highly flexible to support various matching and routing policies.

The basic form of MEDYM is well-suited to service networks with up to thousands of servers. Given that each pub-sub server can support a large number of end users, this scale is adequate for many interesting pub-sub applications in the foreseeable future. For even further scalability, we propose a hierarchy structure called H-MEDYM. Different from existing hierarchal pub-sub network designs, H-MEDYM partitions the server network as well as the content space of a pub-sub system, to effectively reduce server states without introducing skewed load distribution.

The rest of the paper is organized as follows: in Section 2, we briefly review existing pub-sub network design approaches. We present design and efficient implementation techniques of MEDYM in Section 3, and the hierarchy extension to H-MEDYM in Section 4. In Section 5, we present simulation and experimental evaluation results of MEDYM and H-MEDYM, in comparison with the major existing approaches. In Section 5.8, we conclude the paper with directions for future work.

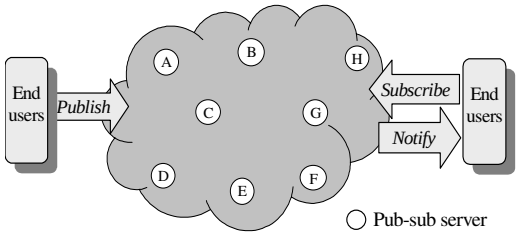

Fig. 1. Example of a publish-subscribe service network.

2 Existing Solutions

Existing distributed content-based pub-sub architecture design can be largely categorized into two classes, which we call the *Content-based Forwarding* (CBF) approach [1][6][7][8][9][21][26] and the *Channelization* approach [11][18][19][25]. They balance the tradeoff between event matching complexity and routing accuracy differently.

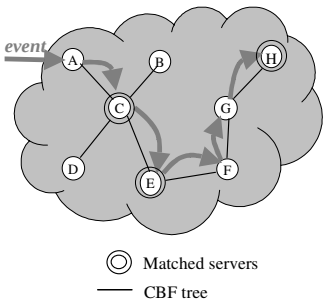

Fig. 2. Event delivery in a CBF tree.

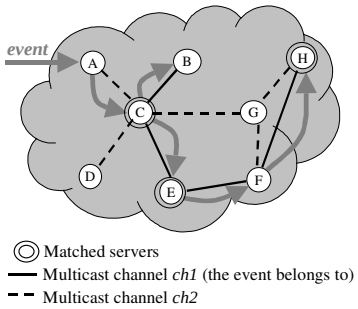

Fig. 3. Event delivery in Channelization.

2.1 Content-based Forwarding (CBF)

CBF proposes an elegant intelligent-network architecture. CBF servers are organized into an overlay network, on top of which one or more *CBF trees* are extracted. For simplicity, we use the single-tree case to illustrate the idea in Fig. 2. Each CBF server maintains a *forwarding table* that keeps track of the sum of subscriptions from servers in each direction of the tree. A published event is broadcast on the tree, matched against the forwarding tables at every step, and forwarded only in directions with matching subscriptions.

Through per-step filtering, CBF achieves highly accurate event routing. Its major challenges are the computation and maintenance cost introduced. First, the per-step content-based event matching is a computationally expensive operation; furthermore, many of the operations may be redundant, as an event may be repeatedly matched with the same subscriptions before reaching the destination servers. Second, events are often routed through uninterested intermediate servers, generating extra network as well as processing load. Servers and network links close to the center of the network are especially likely to carry irrelevant event traffic and become system bottlenecks. Finally, the forwarding tables can be expensive to maintain. When the overlay topology changes, e.g. to adapt to network environment changes, the relative positions of servers in the CBF tree(s) also change. Since subscriptions from each direction on the old tree have been aggregated together in the forwarding tables, there is no easy way to adjust the forwarding tables to reflect the new topology, except by transferring large amount of subscriptions along the new topology and re-computing forwarding tables, generating high network traffic and processing load.

In this paper, we use the work by the Siena group in [6] and [7] as representatives for the CBF approach, as they are perhaps the most prominent and complete works in this direction. They have also designed efficient event forwarding algorithms in [8]. Many other distributed pub-sub systems follow the CBF approach. In JEDI [9], a hierarchical event

routing network was proposed, but was found to perform worse than the peer-to-peer topology in [6]. The Gryphon group [1][26] designed efficient content-based matching algorithms used in forwarding, and proposed using virtual time vectors to convey temporal consistency of subscription propagation. The Elvin system [21] proposed the concept of *quenching*, in which publishers are aware of the sum of all subscriptions in the system, so that they only publish events that have at least some interested subscribers.

2.2 Channelization

The central idea in the Channelization approach is to utilize existing group-based multicast techniques, such as IP multicast or application-level multicast, for event delivery. As shown in Fig. 3, offline, the event space is partitioned into a small number of disjoint event channels. For each channel, a multicast group is built that spans all servers whose subscriptions may match any event in that channel. When an event is published, the server first determines if any server wants the event. If so, it identifies the channel it belongs to, and then sends it to the multicast group for that channel.

The group-based multicast event routing in Channelization is very simple and fast. The main challenge for the approach is its routing accuracy. As discussed in Section 1, event traffic pattern in a content-based pub-sub system is highly diversified. The number of multicast groups a system can build is often much smaller than the total number of different event destination sets. As a result, the same event channel often has to accommodate events with different destination sets, and servers can receive many events that they are not interested in. To reduce such extraneous traffic, intelligent algorithms are used to cluster events with similar destination sets into the same channels. However, the effectiveness of clustering heavily depends on the event and subscription distribution. Unless the distribution offers promising clustering opportunity, as [18] pointed out, it is usually difficult to accurately support diversified user interests with only a small number of groups. Furthermore, the data distribution can be difficult to estimate and change over time.

In this paper, we use [18] as a representative for the Channelization approach. As a companion paper, [19] proposed optimization techniques for [18] and more extensive evaluation results. Although the techniques are proven to be effective, we expect them to be potentially applicable in other approaches as well, and therefore do not consider them as part of Channelization design in this paper. [11] studied the Channelization problem from a theoretical perspective. [25] experimented with different methods of clustering for different data distributions.

3 MEDYM

We propose a pub-sub network architecture called *MEDYM*, for *Match-Early with Dynamic Multicast*. Fig. 4 illustrates the event delivery process in MEDYM: a published event is first matched against subscriptions from remote servers, to obtain a *destination list* of successfully matched servers. Then, the event is routed to these servers through *dynamic multicast*: a transient, stateless multicast tree is computed and constructed on the fly, based on the destination lists carried in event message headers.

MEDYM can be seen as following the *End-to-End* distributed system design principle [20]. It decouples the content-based pub-sub service into two functionalities: complex, application-specific matching at network edge, and simple, generic address-based routing in the network core. Such architecture offers several advantages:

- Low computation cost. Each event is matched with subscriptions for only once; the rest of the delivery process is through simple address-based routing.
- Minimum event traffic. Events are sent only to the servers with matching subscriptions. This not only minimizes the total event traffic on pub-sub servers, but also distributes the traffic consistently with servers' self-interests. Given the heterogeneous user interests in content-based pub-sub networks, this can be an important incentive for servers to join a network.
- Fine-grained routing optimization. Dynamic multicast allows network-efficient routing decisions be made based on individual event traffic patterns.
- Easy deployment and management. Servers are loosely coupled by soft states rather than configured overlay topology. This makes the network easy to deploy and adapt to changes and failures. Content-independent dynamic multicast can also support seamless integration of servers or networks running different pub-sub applications, and upgrade to different data types or matching semantics.

In this paper, we treat the relatively well-studied event matching problem ([1][8]) as an independent plug-in module and do not discuss it further. Next, we present design and implementation of dynamic multicast, and MEDYM server states maintenance. Due to space limitation, we omit the bootstrapping and self-organization of the MEDYM network, which are described in detail in [5].

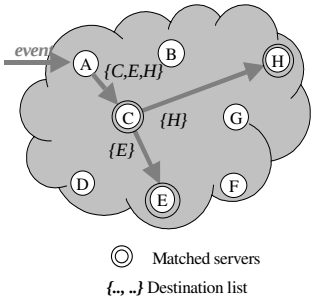

Fig. 4. Event delivery in MEDYM.

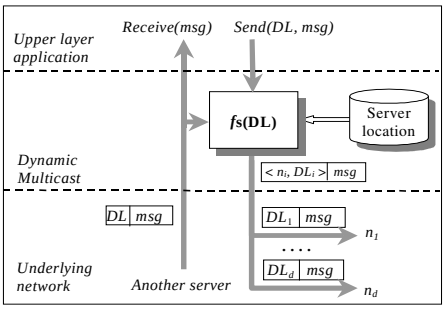

Fig. 5. Dynamic multicast routing.

3.1 Dynamic Multicast

Dynamic multicast is a generic scheme for routing messages to dynamic destination sets. As shown in Fig. 5, it serves a simple interface to the upper layer application: *send* (*DestinationList, message*), and delivers received messages to the application through a callback function *Receive*(*message*). Upon receiving a message with destination list *DL*, from either upper layer application or a remote server, a routing algorithm f_s runs as follows:

$$< n_i, DL_i > = f_s\,(DL) \qquad i = 1 \ldots d$$

The algorithm computes a list of $d < n_i, DL_i >$ pairs, where n_i is the ith next-hop server, and DL_i is the new destination list for n_i. Different routing algorithms can be designed to suit different optimization goals, but the input and output of f_s should always satisfy the following *routing invariants*:

(a) $\bigcup_{i=1}^{d} DL_i = DL - \{s\}$

(b) $DL_i \bigcap DL_j = \phi, i \neq j$

(c) $n_i \in DL_i$

These invariants guarantee that step by step, the message is sent to all its destination servers and to each server only once. Routing loops and redundant paths are naturally prevented. A multicast tree is thus resolved in a recursive way.

One of the major advantages of dynamic multicast is that because no routing states or pre-defined "groups" are maintained, there is no scalability limit on the number of destination sets it can support.

3.1.1 Distributed Dynamic Multicast

To avoid the fragility of centralized decision-making, in this paper, we focus on *distributed dynamic multicast*: each server accurately computes its local part of the multicast tree – its next-hop servers; it resolves the remote part of the tree only on a coarse-grain level, by assigning destinations to the destination lists for the next-hop servers. How the message will be routed beyond the next-hops is transparent and of no concern to the current server. This strategy suits well the fact that servers in a distributed network often have more accurate or up-to-date knowledge about their local environment than distant areas. In the event delivery process, servers improve routing decisions on a finer-grained level, and can easily adapt to network changes or failures. For example, when a server fails to deliver a message to a next-hop server n_i, it simply re-runs $f_s(DL_i - \{n_i\})$ so that the message is still delivered to other servers in DL_i. It also inserts n_i into the destination lists for one of the new next-hops, so that some other server will try to contact n_i, to bypass the possible network failure between the current server and n_i. After three such attempts, n_i is concluded to have failed.

3.1.2 Routing Algorithms

In this paper, we measure communication cost by network latency. Each MEDYM server maintains a *DistanceMatrix,* which contains the latency between every pair of servers in the system. Maintenance of the *DistanceMatrix* will be described in Section 3.2.

To minimize total network cost, we first experimented with routing algorithm that computes the multicast tree as a minimum spanning tree (*MST*) across destination servers. The major drawback of the MST algorithm is its high computation complexity, $O(D^2 \log D)$ where D is the number of destination servers. As the routing algorithm is run in real-time for every event message received, it is important that it can run fast enough to support high event routing throughput.

We then developed algorithm *SPMST,* for *Short-Path-MST*, which computes an *approximate* minimum spanning tree among the destination servers in a fast and distributed way. This algorithm is as shown in Fig. 6. Offline, an array called *ShadowBitVectors* is maintained to help quickly identify next-hop servers. We say that server s_i is *shadowed* by server s_j, if s_i is closer to s_j than to current server s, and s is closer to s_j than to s_i. Under this condition, s_j would forward the message to s_i at lower (latency) cost than s does. Therefore,

a server is a next-hop server if and only if it is not shadowed by any other destination. This can be quickly determined by the intersection of its *ShadowBitVector* and *DLBitVector*, the bit vector for *DL*. After choosing next-hops, the rest destinations are assigned to the destination lists of the next-hop servers closest to them.

Table 1 shows the computation time of MST and SPMST algorithm. The algorithms are written in Java and run with 2.0 GHz Pentium-III CPU and 512MB memory. Results show that SPMST runs much faster than MST, and can support routing of more than thousands of events per second. Furthermore, note that the *average* destination list size in a dynamic multicast message, as analyzed in Section 3.1.3, is much shorter than the |DL| sizes in the table. Therefore, compared to the results on content-based matching [1][8], we expect the computation cost of dynamic multicast routing to be lower and the process faster.

```
computeShadowBitVectors_s() {              // offline
   foreach server s_i {
      foreach server s_j
         if (DistanceMatrix[i][j]<DistanceMatrix[s][i] &&
            DistanceMatrix[s][j]<DistanceMatrix[s][i])
            Set_jth_bit_in_ShadowBitVector[i]; }}

SPMSTRouting_s(DL) {                                  // online
   Nexthops = DL;
   foreach server s_i in DL
      if (ShadowBitVector[i] & DLBitVector !=0)
         Nexthops_remove(s_i);
   if (|Nexthops|>maxNextHops))
      Nexthops = closest_nexthops(maxNextHops);
   foreach server s_j in (DL-Nexthops) {
      n_i = closest_nexthop_to(s_j);
      DL_i +={s_j}; }
   return(<n_i, DL_i>); }
```

Fig. 6. SPMST routing algorithm.

Table 1. Computation time of d-cast routing algorithms, with destination list size |DL|**.**

Routing algorithm	Computation time (ms)		
	|DL|=100	|DL|=500	|DL|=1,000
MST	1.8	9	34
SPMST	0.08	0.29	0.62

Route Caching. An interesting question is whether dynamic multicast routes can/should be cached, so that future routing decisions can be made by cache look-up rather than real-time computation. The effectiveness of caching highly depends on the temporal locality of the pub-sub communication. We plan to study it in the context of specific pub-sub workload in the future, and do not assume caching as a general solution here. This results in a conservative estimation of the dynamic multicast routing computation overhead.

Routing on Mesh. Routing algorithms described above assume that every pair of servers may directly connect, which we expect is the normal scenario in a large-scale dedicated service network. When this is not the case, e.g. due to configurations or network failures, it may be inevitable that event messages be sent through non-destination servers. Our experiments show that the better connected the servers are, the more routing flexibility dynamic multicast can exploit, and the better performance it achieves. As this does not affect

the overall MEDYM design, due to space limitation, we do not discuss such scenarios further in this paper.

3.1.3 Destination List Overhead

Destination lists carried in event messages introduce traffic overhead in MEDYM. Fig. 7 gives an informal analysis of the average destination list size in the process of delivery of one event: in a dynamic multicast tree, the destination lists are reduced at every step by a factor of the fan-out of the server in the tree. Therefore, the average list size is about equal to the diameter of the tree. This is confirmed by Fig. 8, which shows that as the diameters of the SPMST multicast trees are short and grow slowly with total number of destinations, so do the average destination list sizes. For example, to route an event to 1000 servers, an average message carries only 8 server IDs in its destination list. Such overhead is quite acceptable, especially considering that event messages in content-based pub-sub networks often carry rich content, such as attribute-value pairs, full-text or XML documents.

Although the destination lists are short on average, the overhead may not be well balanced, as the lists are longer at locations closer to the publisher. Instead of considering destination lists alone, we developed a routing algorithm to balance server routing load as a whole, as described in [5]. In Section 5, we examine through simulations the destination list overhead in various scenarios.

Note that the low destination list overhead is of critical importance to the scalability of dynamic multicast. [2] and [13] also proposed routing messages based on the destination information carried in message headers. However, as these approaches route messages on top of off-line maintained unicast routes, messages are inevitably sent through non-destination nodes. Traversing such nodes cannot reduce the destination information in the messages. Therefore, the average destination information in the messages is about linear to the number of destinations (rather than about logarithmic in dynamic multicast), and both approaches were developed on the assumption of a very small number (tens of) of receivers.

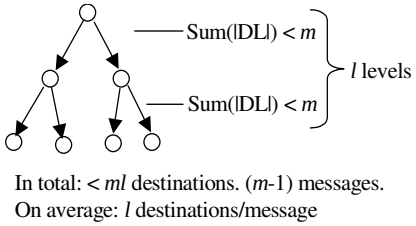

Fig. 7. Intuitive analysis of average destination list size in a dynamic multicast tree.

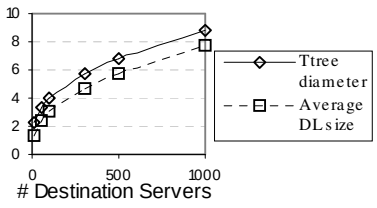

Fig. 8. Simulation results for SPMST multicast tree diameters and average destination list sizes.

3.2 Server States

MEDYM servers maintain two data structures: *routing tables* to support dynamic multicast routing, and *matching tables* for early event matching.

Routing Table. A routing table includes a *server list* of (*serverID*, *IPaddress*, *status*) for all servers in the system, and a *distance matrix M*, where $M_{i,j}$ represents communication cost between server *i* and *j*. In MEDYM, servers periodically broadcast *Refresh* messages

using dynamic multicast. A refresh message contains the server's ID, IP address, network location and status (e.g. load condition). Servers receiving the Refresh message update their routing tables accordingly.

Server network location can be measured in two ways: servers may actively probe each other and broadcast the probing results. This approach generates O(*#servers*^{3}) total network traffic and therefore only scales to small networks. As an alternative, MEDYM can utilize state-of-the-art techniques [16][22] to approximately *estimate* server locations with much lower overhead. Note that inaccurate server location information or even inconsistent information across servers does not affect the correctness of dynamic multicast, which is guaranteed by the routing invariants. In Section 5, we present experimental results of using both probing and the available GNP estimation service [16]. More detailed simulation results of using [22] can be found in [5].

Matching Table. In a pub-sub network, servers often specialize in publishing only certain kinds of events. In MEDYM, a server maintains a matching table, with an entry (*serverID*, *sum_of_subscriptions*) for every other server in the system, which records the sum of subscriptions from that server that are relevant to local publication interest. To make a new subscription or to cancel a previous one (so-called *unsubscibe*), a server broadcasts a *Subscribe* or *Unsubscribe* message via dynamic multicast; servers receiving the message update their matching tables. As an optimization, servers may first broadcast *advertisements* on their publication interests, so that subscriptions are sent (via dynamic multicast) only to servers with relevant advertisements.

Scalability. We do not expect MEDYM routing tables or matching tables to introduce major storage or maintenance overhead for small or medium scale pub-sub networks. First, as each pub-sub server is expected to support a large number of end users, routing tables are expected to be much smaller and more stable than the matching tables. Second, for any pub-sub network to achieve the highly desirable *quenching* capability ([21]), i.e. to filter off events that nobody wants locally, publication servers must know the sum of all (relevant) subscriptions in the network. Servers in MEDYM and the two existing approaches discussed in Section 2 all have the quenching capability, though they differ in subscription replication formats and optimization techniques. We compare their subscription replication cost in detail in Section 5.3.

4 H-MEDYM

MEDYM requires servers to know about all other servers in the network and the sum of their subscriptions. We believe such information needs are practical for service networks with up to thousands of servers. Beyond this point, the storage and maintenance cost of the server states can become the system scalability bottleneck. Hierarchy is an effective method that makes IP routing extremely scalable. However, different from IP addresses, content-based subscriptions from servers geographically close are not necessarily similar and may not be succinctly summarized. Therefore, a similar hierarchy structure for pub-sub network can impose heavy load on servers at the upper level of the hierarchy [6][9].

Based on our experience from an earlier work [4], we propose a different hierarchical solution called for MEDYM, called *H-MEDYM* for *Hierarchical MEDYM*. An H-MEDYM network is partitioned along two dimensions: geographically, servers are clus-

tered based on their network locations; content-wise, the event space is partitioned into non-overlapping *topics*. Each event falls into one topic, while a subscription may overlap with multiple topics (Event space partitioning will be discussed later in more detail). In each cluster, for each topic, one or more servers are designated as *matchers*, which will be responsible for matching events falling into that topic.

An example of event delivery in H-MEDYM is shown in Fig. 9. When an event is published, the publication server identifies the topic the event belongs to, and sends it the closest matcher for that topic in local cluster. At the matcher, the event is matched against subscriptions for that topic, and then dynamic-multicast to two sets of destinations: matched servers in the local cluster, and matched matchers for that topic from remote clusters. At each remote matcher, the event is matched again with subscriptions from servers in that matcher's cluster and dynamic-multicast to the matched servers.

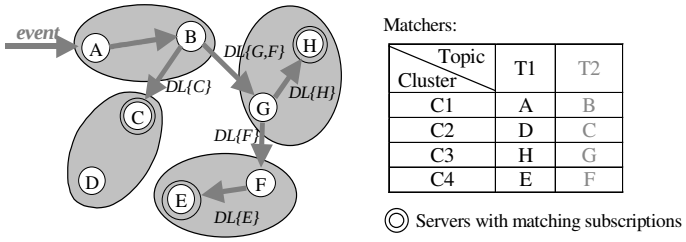

Fig. 9. Event delivery in H-MEDYM. Server *A* publishes an event in topic *T2*. It sends the event to matcher *B*, which matches the event and dynamic-multicasts it to matched servers in the local cluster (not shown) as well as to matched matchers *C, G, F* in other clusters. The event is matched again at *C, G, F* and dynamic-multicast to the matched servers in their local clusters.

4.1 Cluster Configuration and Server States

Unlike in MEDYM, servers in H-MEDYM need to know about only a subset of other servers and subscriptions. Specifically, the first row in Table 2 describes the content of the routing and matching tables at a server that is a matcher: the routing table contains only servers in the same cluster and other matchers for the same topic; the matching table contains only subscriptions from this subset of servers that overlap with that topic (subscriptions overlapping with multiple topics can be divided into smaller subscriptions each covering one topic). A server that is not a matcher (not shown in Table 2) maintains a routing table only for servers in the same cluster, and no matching table.

The second row in Table 2 provides an approximate estimate of the table sizes, normalized as a fraction of global information, i.e. it shows the fraction of all servers or subscriptions in the system that a server needs to know about. The results can be intuitively explained as follows: increasing the number of topics T partitions the event space at finer level, and reduces the number of subscriptions that need to be replicated for each topic and the number of matchers for each topic. However, increasing T beyond N/C no longer reduces matching table size at an average server, as the server now has to match for more than one topic; on the other hand, it continues to increase the routing table size, as the server needs to know about more other matchers. We expect a good H-MEDYM configuration to be around $T \sim N/C \sim N^{1/2}$, in which case the matching table size is reduced by a factor of $O(N^{1/2})$ compared to that in MEDYM. (We do not focus on routing tables, as they

are usually much smaller and more stable than matching tables, as mentioned in Section 3.2). Such reduction can be quite substantial in a large-scale network. In Section 5.2, we present simulation results of H-MEDYM server states under various configurations.

Table 2. Server states at an H-MEDYM matcher for topic *t* in cluster *c*. *N:* #servers, *C*: #clusters, *T*: #topics.

	Routing table	Matching table
Table content	Network location of servers in *c* and matchers for *t*	Subscriptions in topic *t* from servers in *c* and matchers for *t*
Table size, as a fraction of global knowledge	~ $1/C+\max(1/T, C/N)$	~ $\max(C/N, 1/T)$

4.2 Scalability Analysis

Compared to MEDYM, H-MEDYM improves scalability in several aspects. First, it reduces server states as described above. Second, event delivery is divided into two steps: dynamic multicast within each server cluster, and among matchers for the same topic. As each step involves only a small subset of servers, messages carry shorter destination lists. Third, events are no longer matched at publication servers. Separation of publication and matching responsibility allows for more flexible load management, as replication of subscriptions and the workload of event matching can now be allocated based on server capabilities rather than determined by the publication interests of their nearby end users.

On the other hand, H-MEDYM introduces new overheads. An event is now matched twice, at local and remote matchers, before reaching a destination server. The event may also traverse matchers that are not be interested in it. The quenching capability is moved from publication server to the first matcher the event is sent to. Finally, managing server clusters and content space partitions introduces additional cost.

Overall, H-MEDYM trades off efficiency in event delivery for lower server states overhead, and is applicable to very large pub-sub networks where such overhead is the scalability bottleneck. We will evaluate these tradeoffs quantitatively in Section 5.

4.3 Other Issues

Several orthogonal design and algorithmic issues need to be addressed in building an H-MEDYM system. This paper does not make new contributions in these areas. Instead, we explore the possibility of applying existing technologies, and expect these issues to be fertile ground for further optimization and evaluation in the future.

Event Space Partitioning. In H-MEDYM, it is desirable that event space be partitioned into topics with balanced load, and with few subscriptions overlapping with more than one topic. The partition should also be easy to maintain and adaptive to data changes. Many pub-sub applications have inherent concepts of topics, such as news categories, stock industries or geographic area partitions, which are natural candidates for the partitioning in H-MEDYM. In [4] we propose to partition the space into continuous zones, possibly using multi-dimensional partitioning techniques [12][17][24]. Event clustering in Channelization

is also an alternative, although the process can be relatively complex and results sensitive to data distribution. Note that unlike Channelization, H-MEDYM matches each event accurately with user subscriptions, and delivers events only to matched servers or matchers. A bad event space partition is likely to affect the load distribution and server states reduction in H-MEDYM, but will not lead to high extraneous event traffic.

Matcher Selection. When assigning H-MEDYM servers as matchers for topics, subscription replication and the workload of event matching should be allocated consistently with server capabilities. Load distribution is a well-studied problem in parallel and distributed computing [10], and even in pub-sub itself [24]. In H-MEDYM, locality is another key issue: it is desirable that servers match for events that are of local publication or subscription interests, so as to reduce the probability of sending events to matchers who are not interested in them. How to best assign matchers given these potentially conflicting goals is an interesting area for future work.

5 Evaluation

We evaluate our work and compare with existing solutions through qualitative analysis, quantitative simulations and real-world experiments.

5.1 Simulation Methodology

We developed a message-level, event-driven pub-sub network simulator. The IP topology is generated using the GT-ITM transit-stub model [3] with 2500 routers and 8938 links in total. 1000 pub-sub servers are randomly attached to the routers. Each event message has a payload of 200 bytes and a TCP/IP header of 44 bytes. MEDYM/H-MEDYM destination lists have server IDs of 2 bytes each. For simplicity and without loss of generality, we use integers as event and subscription values and perform only equality matching. The results presented are independent of data types or matching algorithms used.

We compare five architectural approaches: MEDYM and H-MEDYM with the SPMST routing algorithm; two versions of CBF: CBF_MST as in [6], where a single CBF tree is built as the minimum spanning tree across all servers, and CBF_SPT as in [7], where CBF trees are shortest path trees rooted at publication servers; Channelization approach as in [18], using Forgy K-Means algorithm to cluster events into 50 channels, as this algorithm was found to produce the best partition results in the paper.

A major challenge in evaluation of pub-sub systems has been the lack of representative application data. In the absence of this, we attempt to gain a comprehensive understanding of the performance of different systems under various distinguishing scenarios. We define a key parameter, *matching ratio*, as the fraction of servers with matching subscriptions for an event, or equivalently, the fraction of events that a server's subscriptions match. We examine scenarios with widely varying matching ratios and our results can be interpreted in several ways: first, low matching ratios imply highly selective subscriptions and high matching ratios represent popular events. We are interested to see how systems perform for these different scenarios. Second, for a given matching ratio, we can understand performance results not only in "absolute" terms, e.g. resource usage numbers, but also in "relative" terms, i.e. how far is the performance from the optimal case. For example, with 10%

matching ratio, a server that receives 20% of all published events can be seen as carrying 100% traffic overhead. Third, a pub-sub network may scale along three dimensions: number of servers, number of total user subscriptions, and volume of event publications. As we focus on evaluation of per event delivery, we do not consider the third factor in this paper. Table 3 shows three scaling scenarios as combinations of the first two dimensions. Users can infer system scalability in these three scenarios from results with different matching ratios. Finally, we have experimented with different event and subscription data distributions, such as uniform, Zipf, exponential and normal distributions. We found that only Channelization is sensitive to data distribution; its clustering is more effective when both event and subscription distributions are highly skewed and have the same peaks. Even so, in all realistic settings, the relative positions of different approaches are the same under all distributions. Due to space limitations, we present results only with uniformly randomly generated event and subscription values, as this provides the most basic and clear understanding of system's performance. Results for many other distributions can in fact be computed as the weighted-sums of the results with different matching ratios.

Table 3. Pub-sub network scaling scenarios.

Scenario	Total subscriptions	Number of servers	Matching ratio
A	↑	–	↑
B	↑	↑	–
C	–	↑	↓

5.2 H-MEDYM Configuration

We first look at the configuration of H-MEDYM networks. We use the Hierarchical Agglomerate Clustering (HAC) algorithm [14] to cluster servers based on their network locations, and partition the event space into continuous ranges with equal lengths. Fig. 10 shows the average size of routing tables and matching tables at H-MEDYM servers, normalized as a percentage of global information (see Table 2). The results validate our quantitative analysis in Section 4.1. Partitioning in both dimensions is necessary to reduce server state in H-MEDYM: when there is only 1 topic or 1 server per cluster (i.e. 1000 clusters), servers maintain 100% of global information. Increasing the number of topics is effective in reducing server states only when there are more servers in each cluster than the number of topics. In subsequent simulations, we use the configuration in Fig. 10 that is closest to our choice of $T \sim N/C \sim N^{1/2}$ in Section 4.1: 20 topics and 50 clusters. In this case, on average, each server knows about 10% of other servers, and 5% of total subscriptions.

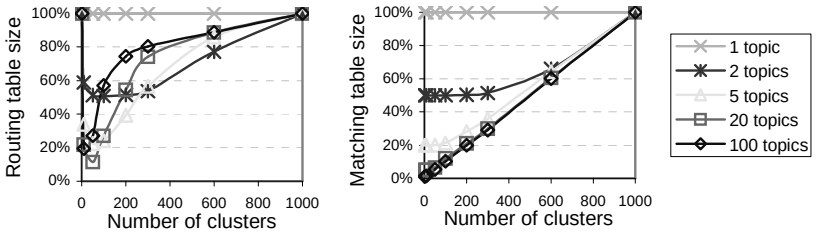

Fig. 10. H-MEDYM server states under different configurations.

5.3 Subscription Replication

Subscription replication is a major source of storage and maintenance cost in pub-sub networks. This cost differs across systems in three ways:

First, the number of remote subscriptions a server needs to replicate depends on its matching responsibility. In MEDYM and Channelization, servers only match for locally published events, and therefore only need to replicate subscriptions that are relevant to their publication interests. In H-MEDYM, a server replicates subscriptions that fall into the topics it matches for, and the replication is independent of its own publication interest. In CBF, a server needs to replicate all subscriptions for which it appears on the CBF tree path between the subscriber and any possibly matching publisher. The number of such subscriptions is dependent on other servers' publication and subscription interests, not just its own.

Second, replicated subscriptions can be *aggregated* ([23][26]) to achieve more efficient storage and update. In MEDYM and H-MEDYM matching tables, only subscriptions from the same server can be aggregated. In CBF forwarding tables, subscriptions from all servers in the same direction in the CBF tree can be aggregated, since the server needs only determine in which directions to forward an event. In Channelization, all subscriptions in the network can be aggregated, as the publication server only needs to know whether an event matches any subscription in the system, for the purpose of quenching. Therefore, Channelization offers greater opportunity for optimization by aggregation than CBF, which in turn offers greater opportunity than MEDYM and H-MEDYM.

Subscription aggregation is a difficult problem whose solution and effectiveness heavily depends on pub-sub data type and distribution. It can also make canceling subscriptions difficult, as mentioned in [7]. Therefore, in Fig. 11, we look at subscription replication assuming no aggregation. In the figure, the x-axis shows an average server's publication interest, measured as a percentage of the entire event space; the y-axis shows the number of subscriptions replicated on an average server, measured as a percentage of all subscriptions in the system. For the CBF approach, we show two curves that represent different subscription selectivity: an average server subscribes to events falling into 1% or 100% of the event space. For example, if each server publishes 1% of events and subscribes to (not necessarily the same) 1% of events, an average server needs to replicate 8% of total subscriptions. This figure can be combined with the effectiveness of a particular subscription aggregation scenario to estimate the subscription replication storage cost in a system.

In the face of network changes, subscription replication in CBF is likely to be more expensive to maintain than in other systems, as discussed in Section 2.1.

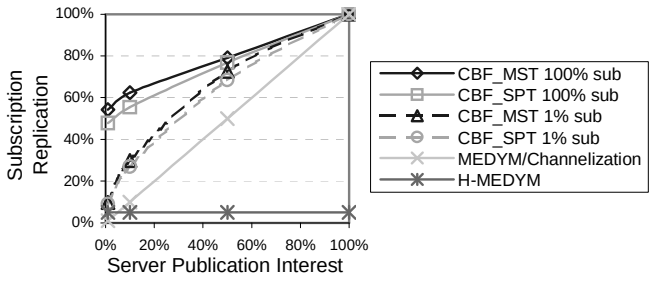

Fig. 11. Subscription replication, without aggregation.

Overall, we expect H-MEDYM to be an effective way to reduce subscription replication, while a quantitative comparison between the other approaches is likely to be dependent on application properties.

5.4 Server Processing Load

For generality and comparability, we measure the processing load at a server by the number of events the server receives. Note that to route each event, the content-based forwarding process in CBF is likely to be more computationally expensive than the address-based routing in Channelization and MEDYM/H-MEDYM, though the concrete results depend on the data type, subscription size, and matching algorithms used. Fig. 12 plots the number of events a server receives, as a percentage of all events published in the system, under varying matching ratios. Channelization servers receive the most events, showing the ineffectiveness of clustering in filtering out extraneous event traffic. When matching ratio is higher than 15%, almost every Channelization server joins all the multicast groups and receives all the events. CBF servers receive much fewer events, due to its accurate per-step filtering. MEDYM servers receive the fewest possible events, i.e. only the events that they subscribe to. H-MEDYM introduces a small overhead over MEDYM, as events can be sent to irrelevant matchers. The difference between the approaches is most apparent when matching ratio is low. For example, a server that subscribes to only 1% events receives 1% events in MEDYM, 2% in H-MEDYM, 8% in CBF_MST, 9% in CBF_SPT, and 29% in Channelization. For very high matching ratios, all approaches converge to broadcast.

Next, we look at the distribution of processing load across pub-sub servers. Fig. 13 shows the percentage of servers that receive no more than a given number of events, when each server matches 10% of total events. We see that all MEDYM servers receive 10% events each, while most Channelization servers receive more than 90% of total events. CBF server load is in between, but is highly imbalanced: about 40% of servers receive only 10% events each, while 20% servers in CBF_MST and 10% in CBF_SPT receive more than 80% events each. As expected in Section 2.1, these heavily loaded servers are located at the center of the network, and route for many irrelevant servers. The imbalance problem is more serious in CBF_MST than in CBF_SPT, because CBF_SPT has multiple CBF trees and higher routing diversity.

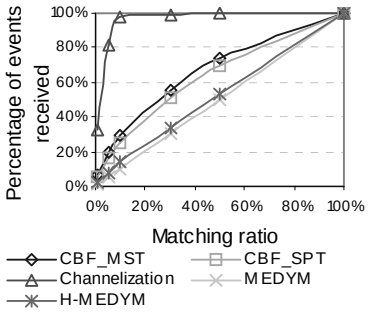

Fig. 12. Average server processing load.

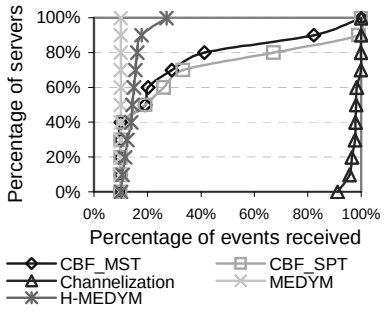

Fig. 13. Cumulative distribution of server processing load, with 10% matching ratio.

5.5 Server Bandwidth Consumption

Server bandwidth is a very precious resource in service networks. Fig. 14 shows the average bandwidth a server consumes in the process of delivering one event. Different from Fig. 12, MEDYM servers only achieve *close to* minimum bandwidth consumption; the difference between its curve and the optimal line shows its destination list overhead. While the overhead is small for low matching ratios, for high large matching ratios (above 90%) it makes MEDYM server bandwidth surpass that of CBF and Channelization by a small amount. H-MEDYM server bandwidth consumption is higher than MEDYM when matching ratio is low, due to events traversing irrelevant matchers, but it is lower than MEDYM when matching ratio is high, due to its shorter destination lists.

Unlike the average case, the maximum server bandwidth consumption can be sensitive to publisher distribution. In Fig. 15 and Fig. 16, we study two extreme scenarios: the *all-publisher* scenario, in which every server publishes the same number of events, and the *single-publisher* scenario, in which only one server publishes all the events. In both cases, the maximum bandwidth consumption in both CBF approaches is much higher than the average consumption, again showing the load imbalance across servers. Different form the processing load case, CBF_SPT has more serious bandwidth imbalance than CBF_MST. This is because the CBF trees in CBF_SPT, built as shortest-path trees on the overlay layer, are likely to degenerate into star-shaped topology with the publication servers at the center (since the shortest path between a publisher and a subscriber is usually just the direct overlay connection between them). Therefore, the publication servers often send out a large number of copies of the same event, and the event routing becomes close to unicasting. The poor performance of CBF_SPT for single-publisher case especially illustrates this point. In MEDYM and H-MEDYM, server load is well balanced; the destination list overhead does not prevent them from significantly outperforming the other approaches for the all-publisher case. However, MEDYM performs less well for the single-publisher case, especially when matching ratio is high, due to the destination list overhead at the publication server. H-MEDYM effectively alleviates this problem, because even with a single publication server, the destination lists are first generated at different matchers. Interested readers can refer to [5] for a dynamic multicast routing algorithm we developed to balance MEDYM server bandwidth.

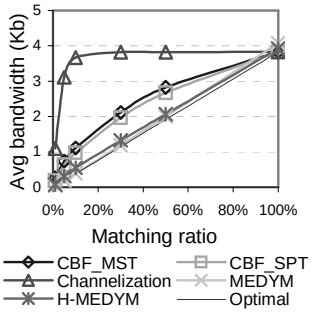

Fig. 14. Average server bandwidth consumption.

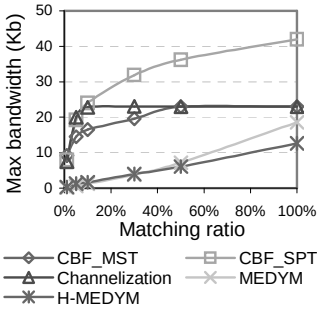

Fig. 15. Maximum server bandwidth consumption in all-publisher case.

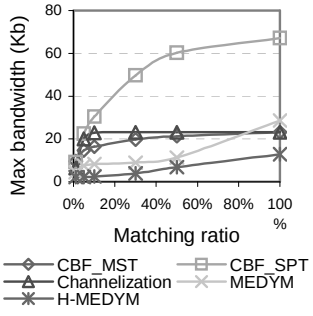

Fig. 16. Maximum server bandwidth consumption in single-publisher case.

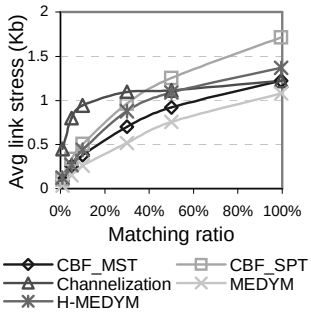

Fig. 17. Average link stress.

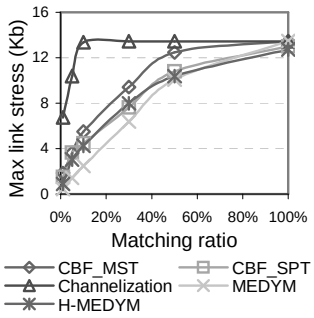

Fig. 18. Maximum link stress in all-publisher case.

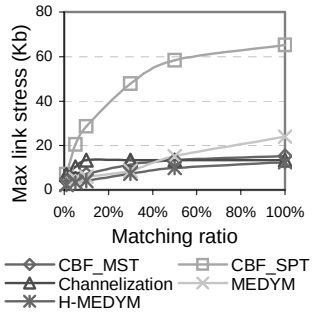

Fig. 19. Maximum link stress in single-publisher case.

5.6 Network Link Stress

Next, we look at event traffic load on underlying network links. We measure *link stress* by the total amount of data transferred over a link in the process of delivering one event. The average link stress results are shown in Fig. 17, and maximum link stress under both all-publisher and single-publisher scenarios are shown in Fig. 18 and Fig. 19. The results exhibit similar trends as server bandwidth consumption results, but the differences between different approaches are of less extent. This is because the underlying IP topology offers lower routing diversity than at the overlay layer: different systems may route events through different sets of servers, but the messages often traverse similar sets of underlying network links, especially when there are only a few long-distance links across IP domains. We expect that in larger IP networks the difference between the approaches would be more significant, and the results would be more favorable to MEDYM and H-MEDYM.

5.7 Event Delivery Latency

In real-time pub-sub applications, it is desirable that events arrive at subscribers within short latency. The end-to-end event delivery latency consists of the processing latency at intermediate servers and the transmission latency on network links. Fig. 20 shows the average number of servers in an event path in different architectures. In CBF, the average event path lengths are always equal to the diameters of the CBF tree(s), with CBF_SPT trees being flatter than CBF_MST trees. In Channelization, when the matching ratio is low, clustering is effective in constructing small multicast groups, and events are routed through fewer servers. In MEDYM, since a multicast tree only spans the matched servers, the average path length is about equal to the logarithm of the number of matched servers. H-MEDYM has shorter event paths than MEDYM, because of the two-level event routing hierarchy. Fig. 21 presents the average Relative Delay Penalty (RDP) of event paths. RDP is defined as the ratio of the sum of network latency of event routing in the pub-sub network over the latency of IP routing between the publication server and the destination server. With shortest path routing trees, CBF_SPT achieves lowest RDP of close to 1. The other approaches all route events along minimum spanning trees or its approximations. RDP in MEDYM and H-MEDYM is lower than in CBF and Channelization, due to the smaller trees MEDYM and H-MEDYM build. H-MEDYM has higher RDP than MEDYM

because events are "detoured" to matchers first. Fig. 20 and Fig. 21 can be used, together with the event processing latency at intermediate servers and the IP latency between publication and destination servers, to estimate the end-to-end delivery for events with certain matching ratios.

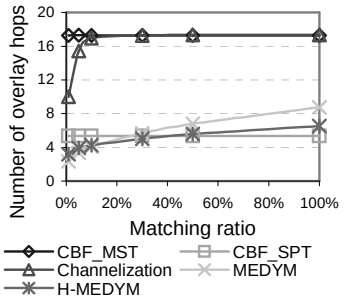

Fig. 20. Average event path length in simulation.

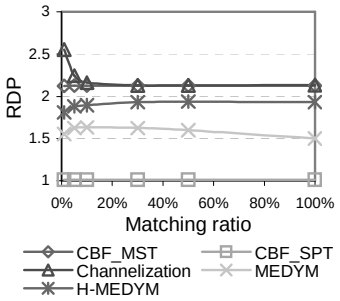

Fig. 21. Relative Delay Penalty (RDP) of event paths in simulation.

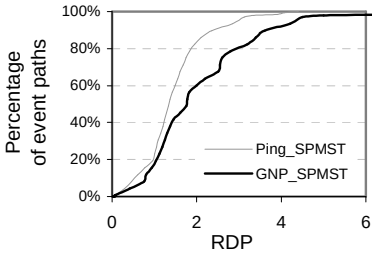

Fig. 22. Cumulative Distribution of RDP in MEDYM deployment.

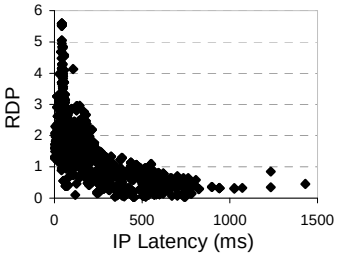

Fig. 23. RDP vs. IP latency using GNP in MEDYM deployment.

5.8 MEDYM Implementation Results

We deployed a prototype of MEDYM on PlanetLab test bed [15]. MEDYM servers are run on 86 PlanetLab sites, 68 in the United States and 18 abroad. Experimental results for server processing load and bandwidth consumption confirm our simulation results above, and are not presented here due to space limitations. To understand event delivery performance in real networks, we focus on network latency results.

In the experiments, we measure server locations in two ways: first, each server randomly pings another server in every 10 seconds, and broadcasts the pinging results every 10 minutes; as an alternative, we used the GNP [16] service to estimate server locations: each MEDYM server pings one of the 8 GNP servers every minute. Based on the pinging results, it computes an 8 dimensional virtual coordinate, and broadcasts its coordinates once every 8 minutes. Distance matrices are then computed locally using the servers' coordinates. Fig. 22 presents the RDP (as defined in Section 5.7) for the event paths. It shows that the routing latency using pair-wise pinging is consistent with our expectation and the overhead of using GNP is quite acceptable. We observe that the inaccuracy of GNP estimation

happens most when servers that are geographically close and derive similar coordinates in GNP in fact have high IP latencies between them, possibly due to congestions or configurations. This can also be seen from Fig. 23, which shows that event paths with high RDP typically have low IP latencies. Overall, Fig. 22 and Fig. 23 confirm our expectation that MEDYM constructs high-quality event routing paths, and network location estimation as by GNP is a promising scalable solution.

6 Conclusions and Future Work

We have presented the design and evaluation of MEDYM, a new architecture for content-based pub-sub service networks, and H-MEDYM, an approach to extend the architecture to a hierarchical structure for greater scale. While these architectures each have their challenges and limitations, we believe that they achieve some important advantages over existing approaches in performance, flexibility and manageability that are highly desirable for many pub-sub applications.

A key goal of our research has been to gain a comprehensive understanding of the characteristics of different content-based pub-sub network designs for different application circumstances. Our evaluation in this paper leads us to the following conclusions.

CBF is an elegant design that achieves accurate event delivery; however, its in-network event processing can be computationally intensive, and the server states that are tightly associated with network topology can be expensive to maintain in a dynamically changing network environment. Therefore, we expect CBF to be suitable for stable pub-sub networks with abundant computational resource. Channelization incurs low computation and subscription replication overhead, but its routing quality heavily depends on pub-sub data distribution and can be very poor when the distributions do not offer very promising clustering opportunity. It is mostly suitable for applications whose user interests can be approximated by a small number of groups with high accuracy.

MEDYM achieves low and well-balanced routing load on servers and network links by sending events only to interested servers via customized routes; its major overhead comes from the servers' global knowledge of location and sum-of-subscription information of all other servers, and the destination lists in its messages. It is well-suited for pub-sub networks with up to a few thousand servers; beyond this point, H-MEDYM is likely to be more suitable: it effectively reduces both the number of servers and the amount of subscription information each server needs to know about, and the destination list overhead. Its overheads are its complexity and the routing constraints it imposes on event delivery paths. MEDYM and H-MEDYM appear to perform well across a range of circumstances; compared to CBF and Channelization, they are most advantageous when user subscriptions are highly selective and diversified. We observe that this is exactly the scenario in which intelligence and efficiency of a pub-sub service is most needed, and therefore their properties would be highly desirable for many applications.

To better understand the characteristics of realistic pub-sub workloads and their implications for architectural tradeoffs, in addition to extrapolating and inferring characteristics from existing information access systems, we plan to deploy a public pub-sub service on PlanetLab [15] and collect real workloads to drive further research. We also plan to investigate several open questions raised in this paper, such as dynamic multicast route caching, event space partitioning and matching distribution in H-MEDYM.

References

1. M. K. Aguilera, R. E. Strom, D. C. Sturman, M.Astley, and T. D. Chandra, “Matching events in a content-based subscription system,” In *Proc. of ACM PODC*, 1999.
2. R. Boivie et al., “Explicit Multicast (Xcast) Basic Specification”, Internet draft, *draft-ooms-xcast-basic-spec-03.txt*.
3. K. Calvert, E. Zegura, and S. Bhattacharjee. “How to Model an Internet-work”. In *Proc. of IEEE INFOCOM*, 1996.
4. F. Cao, J. P. Singh, “Efficient event routing in content-based publish-subscribe service network”. In *Proc. of IEEE INFOCOM* 2004.
5. F. Cao, J. P. Singh, “Towards scalable publish-subscribe service networks”. *Technical Report, Princeton University,* 2005.
6. A. Carzaniga, D. Rosenblum, and A. Wolf, “Design and evaluation of a wide-area event notification service,” In *Proc. of ACM TOCS.* 2001.
7. A. Carzaniga, A.L. Wolf, “A routing scheme for content-based networking". In *Proc. of IEEE INFOCOM 2003*.
8. A. Carzaniga, A.L. Wolf, “Forwarding in a Content-Based Network". In *Proc. of ACM SIGCOMM 2003.*
9. G. Cugola, E. Di Nitto, A. Fuggetta, “The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS”, in *IEEE Transc. on Soft. Eng.*, 2001.
10. D. Culler, J. P. Singh, “Parallel Computer Architecture: A Hardware-Software Approach”, *Morgan Kaufmann*, 1998
11. Z. Ge, M. Adler, J. Kurose, D. Towsley and Steve Zabele, “Channelization problem in large scale data dissemination,” In ICNP, 2001.
12. A. Guttman. “R-Trees: A Dynamic Index Structure for Spatial Searching”. In *Proc. of SIGMOD Conference* 1984
13. C. P. Hall, A. Carzaniga, J. Rose and A. L. Wolf , “A content-based networking protocol for sensor networks”. Tech. Report CU-CS-979-04, University of Colorado, 2004.
14. A.K. Jain, M. N. Murty, and P.J. Flynn, “Data clustering: a review.” In *Proc. of ACM Computing Surveys* 31, 3 (1999), 264—323.
15. PlanetLab Testbed: http://planet-lab.org
16. T. S. E. Ng and H. Zhang. “Predicting Internet Network Distance with Coordinates-Based Approaches.” In *Proc. of IEEE INFOCOM* 2002.
17. S. Ratnasamy, P. Francis, et al. “A Scalable Content-Addressable Network”, In *Proc. of ACM SIGCOMM*, 2001
18. A. Riabov, Z. Liu, J. Wolf, P. Yu and L. Zhang, “Clustering Algorithms for content-based publication-subscription systems,” In *Proc. of ICDCS* 2002.
19. Riabov, Z. Liu, J. Wolf, P. Yu and L. Zhang, “New Algorithms for content-based publication-subscription systems”, In *Proc. of ICDCS* 2003.
20. J. Saltzer, D. Reed, and D. Clark. “End-to-end arguments in system design”. In *ACM Trans. Computer System,* 2(4), pp. 277--88, 1984.
21. B. Segall, D. Arnold. "Elvin has left the building: A publish/subscribe notification service with quenching". In *Proc. of AUUG97*, Brisbane, 1997.
22. L. Tang, M. Crovella. “Virtual Landmark for the Internet”, In *Proc. of ACM SIGCOMM Internet Measurement Conference*, 2003
23. P. Triantafillou, A. Economides. “Subscription summarization: A new paradigm for efficient publish/subscribe systems”. In *Proc. of ICDCS* 2004.
24. Y. Wang, L. Qiu, et. al. “Subscription Partitioning and Routing in Content-based Publish/Subscribe Networks.” In *Proc. of Intl. Symp. on Dist. Comp.* (DISC), 2002.
25. T. Wong, R. Katz, and S. McCanne. “An evaluation of preference clustering in large scale multicast applications,” In *Proc. of IEEE INFOCOM* 2000.
26. Y. Zhao, D. Sturman and S. Bhola, “Subscription propagation in highly-available publish/subscribe middleware”. In *Proc. ACM/IFIP/USENIX Middleware Conference*, 2004.