MEDYM: An Architecture for Content-based Publish-Subscribe Service Networks

Fengyun Cao
Princeton University

Jaswinder Pal Singh
Princeton University

Abstract

Designing a distributed architecture for content-based publish-subscribe service networks is challenging for two reasons: first, communication in such system is guided by the content of publications and subscriptions rather than addresses or network locations; second, while a publication often matches subscriptions from multiple locations, existing group-based multicast techniques such as IP multicast or application-layer multicast are not readily applicable due to the highly diversified patterns of interests. We propose an architectural approach that combines two orthogonal aspects: Match Early, and DYnamic Multicast. We call this a MEDYM architecture, and we compare it with two major existing design approaches: Content-based Forwarding and Channelization. In MEDYM, event publications are matched as close as possible to publishers, to obtain the locations of servers that have matching subscriptions; then, a multicast route to the matching servers is computed dynamically, to best suit the heterogeneous event traffic patterns. We present methods to efficiently implement the dynamic multicast routing. We evaluate the MEDYM architecture using detailed simulations as well as a prototype deployment on the PlanetLab test bed. Experimental results show that the MEDYM approach significantly improves storage, computation and network efficiency compared to existing approaches, and is resilient, scalable, and extensible.

1. INTRODUCTION

Publish-subscribe (pub-sub for short) is an important paradigm for asynchronous communication between entities in a distributed network. In the pub-sub paradigm, subscribers specify their interests as conditions on content of events, and will be notified afterwards of any event generated by a publisher that matches these conditions. Such content-based information delivery is of great value for many distributed applications, such as enterprise activity monitoring, mobile alerting [1], and application integration [13].

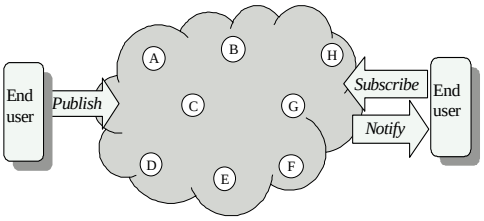
Pub-sub systems can be characterized into three broad types based on the expressiveness of the subscriptions they support. In *channel-based* schemes, events are classified and labeled by publisher as belonging to one of a predefined set of *channels*, to which users may subscribe to. In subject-based or topic-based systems, subscriptions are restricted to fairly narrow conditions on a single dedicated field of an event, usually called the *subject* or *topic* of the event. For example, a finance alert system may use stock ticker as subject of stock price movement events, and a user may subscribe to events with subject "IBM". *Content-based* pub-sub is a more general and powerful paradigm, in which subscribers may specify filtering criteria along multiple dimensions of event content and using complex conditions. A content-based stock alert system, for example, may support subscriptions like "(ticker=IBM) AND (price>100 OR volume>8 million)". Therefore, channel-based and subject-based pub-sub can be seen as special cases of content-based pub-sub.

In this paper, we study the question of architecture design for scalable distributed content-based pub-sub systems that can handle large number of content-based subscriptions and high volume of event publications. At the center of the architecture design is the question of how to efficiently deliver events to all subscribers whose interests they match.

Content-based event delivery is challenging in part because it cannot be directly supported by address-based Internet routing primitives. An event may match the interests of multiple subscribers, but existing group-based multicast techniques such as IP multicast or application-layer multicast [12][4] cannot be directly applied to event routing, because content-based subscriptions are highly diversified: Different events may satisfy the interests of widely varying groups of users. In the worst case, mapping all possible event traffic patterns into multicast groups may require a number of groups exponential in the network size (i.e. 2^n where n is the number of users).

The conceptually simplest architecture for a pub-sub system is a centralized one. There is a single pub-sub "server" that consists of one or multiple machines on the same local area network (LAN). Clients submit their subscriptions and publications to the server, which matches published events with all subscriptions and sends notifications to clients with matching subscriptions. This architecture has several drawbacks: first, unicasting notifications for an event from the server to all interested clients (which may also be far away) is not network efficient. Second, many events that match no subscription in the system may be sent to the server across the network, as there is no way of knowing whether anyone is interested in an event before it is matched. It may also be difficult or uneconomical to find a single location on the Internet with high enough access bandwidth to support the heavy incoming and outgoing traffic with large numbers of publisher and subscriber clients. Finally, the system has a single point of failure: no communication is possible if the pub-sub server location goes down.

To achieve higher scalability and reliability, like other



Server	Subscriptions
A	{1,5}
B	{7,8}
C	{1,2}
D	{0,6}
E	{2,3}
F	{5,7}
G	{4,6}
H	{2,9}

Figure 1. Example of a pub-sub service network. There are 8 servers and each server has 2 subscriptions. Event values range from 0 to 9.

previous research [2][7][8][9][11][13] we focus on architecture design for a distributed pub-sub service network, as shown in Figure 1. A set of pub-sub servers controlled by the service network are widely distributed over the Internet; clients access the pub-sub service, either to publish events or to register subscriptions, through appropriate servers, such as those that are close to them or in the same administrative domains. Thus, pub-sub servers serve as publication/subscription proxies on behalf of clients, and we can view the problem as one of getting events from servers where they are published to the servers that subscribe – as proxies – to the events. We call these publication servers and subscriber servers in the rest of this paper. Note that the same server may serve as a publication server and as a subscription server. Communication between pub-sub servers and their associated clients is a separate matter that is now localized and is not discussed in this paper.

In a distributed pub-sub service network, servers cooperate to *match* events with subscriptions and *route* events to interested subscription servers. The matching and routing problems are interrelated: event routing decisions are based on matching results, and underlying routing capabilities can affect where and how matching is performed. In this paper, we use the term *architecture* to refer to manner in which matching and routing are distributed and coordinated in the system. We discuss how existing distributed pub-sub proposals approach these problems, and propose a new architecture. We evaluate architecture approaches along three dimensions:

1. Performance and resource usage efficiency. Resource consumption in pub-sub systems usually includes subscription storage, matching/routing computation, network communication load and management overhead. It is also important that resource usage be well enough balanced so that total system throughput is not compromised.
2. Service reliability. In a large-scale distributed system, server and network failures and condition changes are considered normal rather than extreme scenarios. Static resilience [17], adaptation to changing environments, and failure recovery are important issues.
3. Scalability and extensibility. Given the fast growing nature of information volume and needs, the pub-sub architecture should scale gracefully to large user population and heavy workload. It should also be extensible to more sophisticated forms of subscriptions, such as composite subscriptions that depend on occurrence of multiple events published from potentially different sources in the network, and new data types and matching functionalities.

The rest of the paper is organized as follows: In the next section, we review two major existing architecture designs for content-based pub-sub service networks and examine their tradeoffs. Based on this analysis, we propose a new architectural approach called Match Early with Dynamic Multicast (MEDYM), in Section 3. Section 4 discusses the MEDYM approach and some efficient implementation mechanisms in greater depth. In Section 5, we evaluate the performance of MEDYEM and the other two major existing approaches using detailed simulations. Since the simulation results endorse the potential of the approach, we plan to implement and deploy a pub-sub service network using the MEDYM approach, to gain experience with it on real systems and workloads. Section 6 describes our experience so far with implementing a prototype MEDYM system on the PlanetLab distributed testbed [2]. Section 7 discusses related work, and Section 8 concludes the paper and discusses directions for future work.

2. EXISTING APPROACHS

Existing content-based pub-sub service network design can be largely categorized into two classes, which we call the *Content-based Forwarding* (CBF) approach [3][9][10][11][13] and the *Channelization* approach [22][23][28].

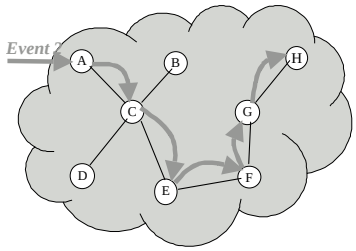
2.1 Content-based Forwarding (CBF)

We use the Siena system [10] as a representative for CBF approach. As shown in Figure 2, servers organize into an overlay network with acyclic (tree) peer-to-peer topology[1,2], which we call a *CBF tree*. Servers broadcast their subscriptions on the tree, and each server records the sum of subscriptions from each neighbor's direction in its *forwarding table*. *Advertisement* is an optional technique to constrain subscription flooding: servers may first broadcast advertisements that describe the possible content of their future publications, so that subscriptions only need to be sent toward servers with matching advertisements, i.e. servers that have the potential to publish events that match these subscriptions. We will discuss effectiveness of advertisements in more detail in Section 5.

When a CBF server receives an event, it matches the event with subscriptions in its forwarding table, and forwards the event to neighbors whose directions have matching subscriptions. Note that an event does not need to be matched against individual subscriptions. Instead, subscriptions in the same forwarding table entry can be aggregated and the

[1] [] proposed that Siena can work with a cyclic network topology by first extracting a routing tree rooted at the origin of the message. However, event routing is still along a pre-configured acyclic topology and therefore is not further discussed in their papers.

[2] Another acyclic topology, i.e. hierarchical topology, was shown to perform worse than the peer-to-peer topology and therefore is not considered in this paper.



Server	Forwarding table	
	Neighbor	Subscriptions
A	C	{0-9}
B	C	{0-7,9}
C	A	{1,5}
	B	{7,8}
	D	{0,6}
	E	{2-7,9}
D	C	{1-9}
E	C	{0-2,5-8}
	F	{2,4-7,9}
F	E	{0-3,5-8}
	G	{2,4,6,9}
G	F	{0-3,5-8}
	H	{2,9}
H	G	{0-8}

Figure 2. Example CBF network structure and event delivery. Forwarding tables are shown on the right.

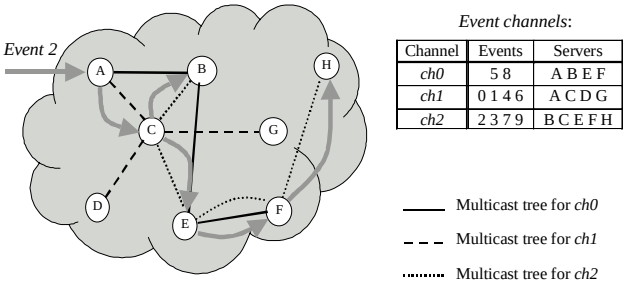


Figure 3. Example Channelization network structure and event delivery. Event space is clustered into three channels using Forgy K-means algorithm.

matching process only need to determine whether a direction matches or not as a whole. Siena has proposed exploiting *covering* and *merging* relationships among subscriptions from multiple servers as they are combined and forwarded in the method, to improve storage and matching efficiency. However, this has not been evaluated and it is an open question as to how efficient and effective the rules are, especially with high dimension event space.

In CBF, events are only forwarded along directions that lead to matching subscriptions. In this way, CBF achieves network efficiency elegantly. However, its operating and maintenance cost can be high for several reasons.

First, content-based matching of an event with subscriptions in the forwarding table is an expensive operation. It includes parsing content of various fields of the event message and performing complex tests, such as full-text search, range queries or Boolean expressions, against large number of subscriptions. The complexity of matching is determined by the nature of the application, usually being more expensive as more flexible and powerful subscriptions are supported. Furthermore, in CBF, many of the matching operations may be redundant. In Figure 2, event *2* is repeatedly matched with subscriptions from server *H* at server *C*, *E*, *F*, *G* before reaching *H*. These matching operations result in high computation load on pub-sub servers.

Second, event routing in CBF is limited by the use of a single tree topology. An event is often routed through intermediate servers that have no subscriptions interested in the event, such as server *F* and *G* in Figure 2, thus introducing unnecessary computation and network traffic load on such servers.

Third, the CBF tree topology has high maintenance cost. Because the content and organization of forwarding tables are tightly coupled with the tree topology, change of servers' relative positions in the topology requires adjustment of subscription content in forwarding tables. This can be difficult because the servers may not know exactly which server each subscription is from, and may have to consult remote servers, resulting in high volume of network traffic.

Finally, the topology leads to load imbalances. Servers and network links located close to the center of the CBF tree are likely to route for much more (irrelevant) events than their peripheral counterparts and become system bottlenecks.

2.2 Channelization

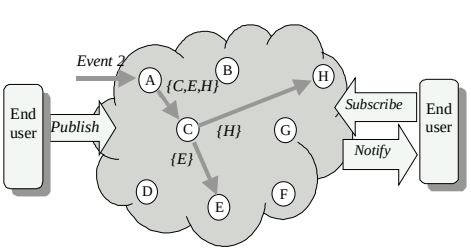
We use [22] as a representative for the Channelization approach. As shown in Figure 3, the multi-dimensional event space is partitioned into a limited number of event channels offline and the partitioning result is replicated on all servers. For each event channel, a multicast tree is built that spans all servers carrying subscriptions that could match any event in that channel. A publication server uses the event space partition to determine which channel the event belongs to. The event message also carries the channelID of the event, so that its content does not need to be matched again at subsequent servers in the multicast tree. Event forwarding is thus simple and fast.

Because the available number of multicast channels is often much less than the 2^n possible event traffic patterns, servers in the same multicast tree may have substantially non-overlapping interests, and hence a server may receive many uniteresting events. In Figure 3, event *2* is sent to five servers while only two of them have matching subscriptions. Especially, server *B* is neither a matching server itself, nor does it lead to any server with matching subscriptions.

To reduce extraneous event traffic, the Channelization approach uses clustering algorithms to partition event space, to maximize the commonality of subscription locations of events in the same channel. However, the effectiveness of clustering heavily depends on the content distribution of events and subscriptions. If the distribution does not lend itself to promising clustering opportunity, it is generally difficult to match the highly diversified user interests in a content-based pub-sub system into a small number of groups with high accuracy. Furthermore, the distribution itself may be difficult to estimate with high accuracy and may change frequently over time, requiring clustering to be recomputed and results updated between servers.

2.3 Discussion

The CBF and Channelization approaches both route events along pre-configured network topologies. Because the available choices are limited *a priori*, the routing paths cannot be optimized based on individual event traffic patterns, and the inefficiency of events traversing irrelevant servers cannot be



Information at a MEDYM server:

Server	Sub-scriptions	Network location
A	{1,5}	…
B	{8,9}	…
C	{1,2}	…
D	{0,6}	…
E	{2,3}	…
F	{5,7}	…
G	{4,6}	…
H	{2,9}	…

Figure 4. Example MEDYM network structure and event delivery. The destination lists are shown in braces.

avoided. Furthermore, maintenance and adaptation of the network topology can be quite expensive in the face of changing subscriptions.

However, the two approaches balance the tradeoff between cost and quality of event delivery differently: CBF performs content-based matching for every event at every forwarding step, while Channelization only approximates destination grouping patterns on a coarse-level, using offline clustering results. As a result, we expect that CBF achieve better routing accuracy at higher operation cost, while Channelization is simpler to perform but may also generate more extraneous network traffic.

3. MEDYM HIGH-LEVEL DESIGN

Based on the above observations, we propose a new architecture for content-based pub-sub service network called MEDYM: *Match Early with DYnamic Multicast*. To avoid high forwarding cost, content-based matching is performed only once in MEDYM, as close to the publication server as possible[3]. The result of the one-time matching is recorded with the event message as a *destination list*; from then on, event forwarding is based on the addresses in the destination list rather than on the content of the event. Therefore, event forwarding is through address-based routing, as opposed to content-based routing, and is expected to be simple and fast. To allow maximum routing flexibility, MEDYM assumes no predetermined routing topology. Rather, routing decisions are dynamically made for each event based on its destination list, and different routing algorithms can be used for different optimization goals.

Figure 4 shows an example MEDYM network. Each MEDYM server maintains two kinds of information for all other servers in the system: the sum of subscriptions on each server, which will be used in matching, and the servers' network locations, which will be used in routing. The server matches locally published events against subscriptions to obtain a destination list of matching server IDs. (The specific content-based matching algorithms used are plug-in modules that are independent of the architecture, and are not a focus of this work.) Once matched, an event always carries a destination list that indicates the destination servers that the current server is responsible for event delivery. Based on the destination list and knowledge of server locations and network conditions, the server implements *dynamic multicast routing*: it computes next-hop servers to which to forward the event, as well as the new destination lists for each of the next-hop servers, and sends the event and (reduced) destination lists to these next-hop servers.

MEDYM routing for the example pub-sub system is shown in Figure 4. After matching at server *A*, event *2* has destination list of the three matching servers {*C, E, H*}. It is first sent to *C*, where it is forwarded to *E* and *H* with new destination lists {*E*} and {*H*}. Because MEDYM does not assume any pre-configured network topology, the event can be routed so that it touches only the three servers that are its destinations.

MEDYM combines key advantages of the CBF and Channelization approaches. It matches every event against subscriptions for high routing accuracy, like CBF, but does this early and avoids repeated matching. Once matched, events are routed fast based on addresses as in Channelization, but with flexibility rather than with constraints on topology or channels.

The major challenge for the MEDYM approach is design and implementation of correct and efficient dynamic multicast scheme. The rest of this section focuses on dynamic multicast.

3.1 Dynamic Multicast

The functionality of dynamic multicast routing at a MEDYM server *s* can be described as:

$$\{n_i, DL_i\} = f_s(DL), \quad i = 1...d$$

- DL: destination list in the event message server *s* receives.
- f_s: multicast routing algorithm at server *s*.
- n_i: the *i*th next-hop server to forward event message to.
- DL_i: destination list for the message sent to server n_i.
- d: number of next-hop servers.

3.1.1 Routing invariants

For routing correctness and efficiency, we define the following *dynamic multicast routing invariants*:

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

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

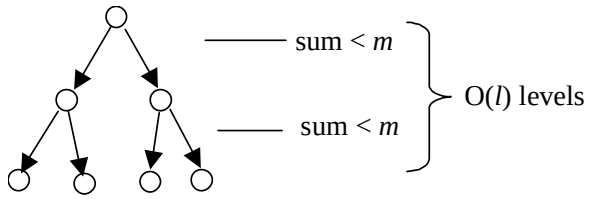
(c) $n_i \in DL_i$

As discussed below, the correctness and efficiency of dynamic multicast as proposed in MEDYM are derived from these routing invariants. Various concrete routing algorithms can be developed for different routing policies, as long as they conform to the interface and invariants above.

3.1.2 Dynamic multicast properties

Several important properties of dynamic multicast can be inferred from the routing invariants, assuming that all servers are reachable and function correctly:

[3] The matching is typically performed at the publication server itself, except when this is altered for load balancing, see later, or when the system is extended to support composite subscriptions, involving events from multiple publication servers, in which case the matching will be performed as close to the publication servers as possible.



In total: < O(*ml*) destinations. (*m*-1) messages.
On average: *l* destinations/message

Figure 5. Analysis of average destination list size in dynamic multicast.

(i) Dynamic multicast delivers an event to all servers with matching subscriptions.

(ii) Dynamic multicast is loop free, i.e. an event traverses a server at most once.

Therefore, routing paths for an event form a tree topology, though the actual routing process is stateless – no information about such multicast tree is stored at pub-sub servers.

(iii) Dynamic multicast route events only through servers that are interested in them.

This property guarantees that the multicast tree for each event spans the minimum set of servers, reducing total computation and network bandwidth usage in the process of event delivery. It also aligns resource consumption with self-interests of servers, as a MEDYM server only receives events that it has interest in, and matches (with remote subscriptions) only events locally published.

(iv) The average destination list size in all messages for an event is O(l) servers, where l is the diameter of the dynamic multicast tree for the event.

Since the destination list From invariant (a) we have

$$\sum_{i=1}^{d} size(DL_i) = size(DL) - 1 < size(DL)$$

As illustrated in Figure 5, at each level of the multicast tree, the sum of destination list sizes in all messages is less than *m*, where *m* is total number of matching servers. Because the tree is of depth *l*, the sum of destination list sizes in all messages is less than *ml*, and there are (*m*-1) messages in total, so that average destination list size is O(*ml*/*m*)=O(*l*). In particular, for a multicast tree with diameter O(log*m*), such as a random tree which is often the shape of multicast tree in practice, the average destination list size for an event is only O(log*m*).

Although the real destination list size depends on the shape of the multicast tree, this property provides a way to approximately estimate the destination list overhead in event messages. For a pub-sub system with 10,000 servers, assume that an event matches interests of 20% of the servers, and the dynamic multicast tree is a random tree, the average destination list is expected to consist of about ln(10,000*20%) = 8 servers; using 32 bit server ID, the destination list length is 8*4 = 32 bytes. This overhead is expected to be acceptable, considering the often rich event content in content-based pub-sub systems, such as in the form of XML messages. Compression and coding techniques may be used to further reduce the destination list length, and is a direction for future work.

We have shown that the average destination list overhead is low. However, the destination list size is longer at publisher than at intermediate forwarding servers. If a server publishes a lot of events that match subscriptions at many servers, the destination list overhead at this server is relatively high. To solve this problem, we introduce the concept of a *helper*. A server chooses a server that matches the event it publishes. It forwards the event to the helper without destination list. The helper then matches the event again and initiates dynamic multicast as if it published the event itself. In this way, the overhead of destination list size is shifted from the original publication server to the helper. By strategically assigning helping relationships, this mechanism helps to balance the destination list overhead across servers.

From another perspective, a server ID in the destination list can be seen as a reasonable "cost" that the publication server "pays" for getting the event delivered to that destination server through the pub-sub system. Dynamic multicast is an *address-based* routing service, and a user is expected to provide the destination(s) of a message. Therefore, we do not see this as a problem load imbalance problem in event routing and do not further investigate it in this paper.

3.1.3 Dynamic multicast methods

Source-based dynamic multicast

A straightforward way to implement dynamic multicast is by source-based routing. The publication server computes the multicast tree for an event and encodes the tree in the event message, probably by ordering the servers in the destination list. A forwarding server decodes the tree and forwards the event as instructed. The drawback of this approach is that the routing quality depends on the knowledge and intelligence available at the publication server at the time of event publication. If the server's knowledge about the service network status is out-of-date or incomplete, the multicast tree computed can be sub-optimal or even incorrect.

Distributed dynamic multicast

We observe that the interface of dynamic multicast routing only requires a server to resolve the local part of the multicast tree. Therefore, dynamic multicast can also be implemented in a distributed way.

In distributed dynamic multicast, no server is responsible for or knows about the entire multicast tree. Each server only determines its next-hop servers and their destination lists; exactly how these next-hop servers are going to deliver the event is out of concern and transparent to the current server.

Distributed dynamic multicast is highly flexible and resilient. Servers may run different routing algorithms to best fit their routing policy, and make real-time routing decision adjustment for condition changes. When a server observes that a next-hop server went down, it can simply re-run the routing algorithm with the failed node removed from the destination list, so that the event can be routed around that server.

In Section 4 we will design concrete routing algorithms for

both source-based and distributed dynamic multicast.

3.2 Discussion

In this Section, we introduced MEDYM, a new architectural design for content-based pub-sub service network. It combines two orthogonal aspects: Match Early, and Dynamic Multicast. We believe that content-based matching for every event is necessary to achieve high routing efficiency and should be performed as early as possible in the process of event delivery. To suit the heterogeneous event traffic pattern, we propose dynamic multicast as a general event routing scheme for distributed content-based pub-sub system. We defined the functionality and invariants for dynamic multicast routing, and analyzed its correctness and efficiency features. We also discussed two ways of implementing dynamic multicast and their tradeoffs. In the next Section, we discuss in detail design and efficient implementation of MEDYM system.

4. MEDYM DESIGN DETAILS

As a new architectural design, MEDYM faces potential challenges that are not present in the existing approaches. In Section 3, we have analyzed the destination list overhead in MEDYM. In this Section, we discuss several other points of concern in MEDYM and present ways to address them in detail.

4.1 Efficient Dynamic Multicast Routing

In dynamic multicast, routing paths are computed for every event in real time. It is important that the routing algorithm is efficient enough to support high volume of event traffic.

In this paper, we develop routing algorithms that aim at optimizing the total network communication cost in event delivery. We use network latency as communication cost between servers, and design three dynamic multicast routing algorithms as follows:

MST is a routing algorithm that computes minimum spanning tree for servers in the destination list. The neighbors of the current server in the minimum spanning tree will be used as next-hop servers, and each sub-tree of the neighbor will be used as the next-hop server's destination list. The algorithm is the same for source-based or distributed multicast. The computation time for *MST* is of $O(D^2 \log D)$ where D is the number of servers in the destination list.

CloseMST is a distributed routing algorithm designed to expedite routing computation by approximation. Instead of computing a minimum spanning tree for all servers in the destination list, a server first chooses the closest c destination servers as *candidates*, and computes a minimum spanning tree for the candidates. Then, it uses the neighbors in this small minimum spanning tree as next-hop servers, and assigns the other destination servers into the destination list of the closest next-hop server to that destination. In this way, the computation complexity of the algorithm is reduced to $O(c^2 \log c + Dc)$. In experiments, c is set to a small constant number around 10, and therefore the computation time is linear to the destination list size. We have also explored alternative ways of choosing the candidate servers, such as by clustering servers by their network locations or directions offline, but their routing qualities are not as good as that of *CloseMST*.

BalanedMST is a variant of *CloseMST* that takes server load into consideration when making routing decisions. Even though dynamic multicast does not route events along one pre-configured tree topology, servers at the center of the network are still more likely to be chosen as intermediate servers in multicast trees and forward more copies of event messages than their peripheral counterparts. *BalanedMST* is the same as *CloseMST*, except that a server computes the minimum spanning tree only for the candidate servers that are not overloaded. If all candidate servers are overloaded, it chooses one closest not overloaded destination server as the only next-hop server. The overloading criteria can be specified by the application. In our experiments, we define a server to be overloaded if the ratio of number of messages it sends to the number of messages it receives is above an *OverloadThreashold (OT)*. When this ratio is high, it indicates that the server has high connectivity degree in multicast trees and consumes much network access bandwidth.

Table 1 presents performance and quality results of the three routing algorithms. The simulation details are as described in Section 5. The algorithms are written in Java and run on PC with 2.0 GHz Pentium-III CPU and 512MB memory. The running time is computed as average of 100,000 times running. Routing quality is measured by the Normalized Resource Usage (NRU), a normalized total network resource usage metric, of the resulting multicast tree.

Table 1 shows that the approximate routing algorithms are significantly faster than *MST*, while achieving comparable routing quality. *BalancedMST* has higher NRU as it sometimes chooses sub-optimal routing paths for load balancing. More detailed results can be found in simulation results in Section 5. In Section 3, we have shown that the average destination list size is usually quite short. Therefore, these routing algorithms, especially the approximate ones, are expected to be efficient enough for large-scale pub-sub systems with high event traffic throughput. The running time of these algorithms are also much shorter than the content-based matching time presented in pub-sub literatures [11][18][24], confirming our expectation that address-based forwarding are faster and cheaper than content-based forwarding.

4.1.1 Caching routing decisions

An interesting question is that whether routing results could be cached at MEDYM servers, so that routing for events with the same destination sets can be solved by a cache

Table 1. Running time and routing quality of dynamic multicast routing algorithms.

Routing algorithm	Configuration	Running time (ms)		Quality (NRU)	
		D=100	D=2000	D=100	D=2000
MST		1.8	144	105%	122%
CloseMST	c=16	0.08	1.12	121%	157%
BalancedMST	c=16, *OT*=5	0.08	1.31	150%	216%

lookup instead of routing algorithm computation. Like in the Channelization approach, effectiveness of such caching depends on the locality property of communication patterns: caching is most effective if and only if some event traffic patterns are more often than the others. We plan to investigate caching mechanisms with realistic application properties in the future.

4.2 Managing and Updating Information

In MEDYM, each server needs to know about other servers' subscriptions for content-based matching, and network locations for dynamic multicast routing. In the most straightforward way, such information can be broadcasted using a multicast group. Below we describe techniques to improve information update efficiency.

4.2.1 Subscriptionse

With use of advertisement, subscriptions in MEDYM can be sent through dynamic multicast to servers with matching advertisements. From *property (iii)* in Section 3, we know that the subscription will be only forwarded to, and stored at, servers with relevant advertisements. Compared to subscription broadcast, this reduces the subscription update traffic and the amount of subscriptions stored and matched at publication servers. The improvement results are shown in Section 5.

4.2.2 Network locations and server status

Dynamic multicast requires each server knows about network locations of all other servers. This is not true in CBF or Channelization, where servers only need to know about a fixed set of neighbors. However, we do not expect this to be a big overhead. First, research works have designed network location estimation techniques [15][20][21] that efficiently estimate and summarize server location information with reasonable accuracy. For example, in GNP [21], servers summarize their network locations using coordinates in a virtual *d*-dimensional Euclidean space. Servers only need to broadcast their coordinates, and locally computes latencies between remote servers based on their coordinates. For a network of 10,000 servers, with each update message of 1000 bits and every server broadcasting every 10 minutes, the bandwidth consumption for network location update is only about 10,000 * 1000/600 = 17k bps. Other server status information such as load conditions can be broadcasted in a similar fashion. Furthermore, we expect that such per-server level network information is of much smaller size and update frequency than the per-client level subscription information and is not the major management overhead.

4.3 Inter-server Communication

Reliable event delivery is an important feature of a pub-sub service. We propose to use reliable network protocol such as TCP for inter-server communication. In dynamic multicast routing, a server does not have a fixed set of neighbors. Instead, it computes the multicast tree assuming an underlying full-mesh (IP) network, and may need to directly forward events to any other server(s) at any time. In case of high event traffic volume, frequently opening and closing TCP connections is expensive and limits the maximum traffic throughput. We address this problem in three ways:

First, TCP connections can be left open between adjacent event transmissions between the same pair of servers for a limited period of time, in a similar fashion as the persistent connection optimization in HTTP/1.1. If the total number of simultaneously open TCP connections is restricted by the system's limit, we may consider switching reliable UDP or user-level TCP [14].

Second, event transmissions between the same pair of servers can be batched together, within an acceptable latency. In this way, connections are open only periodically and only when there are events waiting to be sent.

Third, we may restrict which servers can directly communicate. This is like imposing a mesh overlay topology upon which routing paths can be built. The mesh topology represents the tradeoff between connection management and routing quality. This solution violates the dynamic multicast routing invariant (c) defined in Section 3, and we leave it as a future work.

4.4 Scaling to Very Large Systems

The current MEDYM architecture has a flat, peer-to-peer network structure. This architecture is designed for pub-sub service network with thousands or tens of thousands of servers. Because each server can support a large number of individual clients, we expect such scale is large enough for many pub-sub applications. For a system with even larger scale, we expect that a hierarchical architecture with multiple levels of MEDYM would be a more efficient choice. In such system, servers are organize into clusters, so that information update and routing decisions can be localized. We believe that the detailed design of multi-level MEDYM should be based on full understanding and evaluation of current architecture and therefore leave it as a future work.

5. EVALUATION THROUGH SIMULATION

In this Section, we present evaluation of the three pub-sub service network architectural designs, i.e. CBF, Channelization, and MEDYM, using message level event-based simulation.

5.1 Simulation Setup

5.1.1 Network topology

The network topology we used is generated by GT-ITM random graph generator using the transit-stub model. There are 20 transit domains with an average of 5 routers in each. Each transit router has an average of 3 stub domains attached, and each stub domain has an average of 8 routers. Altogether there are 2500 routers and 8938 links. The link latencies are random numbers between 50-100ms for intra-transit domain links, 10-40ms for transit-stub links, and 1-5ms for intra-stub domain links. Pub-sub servers are randomly attached to the

routers by LAN links with 1ms latency. IP multicast routing is simulated using a shortest path tree formed by merging IP unicast routes from the source to each destination. We have also simulated network topology generated by Inet [19] and from Rocketfuel [25]. The resulting trends are very similar to the ones from GT-ITM and are omitted due to space constraint.

5.1.2 Workloads

A major challenge in pub-sub system evaluation is the lack of real-world workloads. For comprehensiveness, we experiment with four publication/subscription distributions and study their effects on system performance. These distributions either are prevalent in other information delivery applications and/or have been used in pub-sub literatures:

- Uniform distribution, in which both publications and subscriptions are uniformly randomly distributed.
- Zipf-uniform distribution, in which subscriptions follow Zipf distribution [6] with Zipf parameter set to 1.2, and publication is uniformly distributed, as in [29].
- Multimodal distribution, in which publication and subscription distribution follow the same multivariate Gaussian distribution. In this scenario, events that are popular are also published more often. In our experiments, five distribution peaks are randomly chosen in the event space, and the standard deviations are set to 1/4 of the distances between peaks.
- Regional distribution [28][22], in which the probability that a subscription from server s_i matches an event from server s_j is set to:

$$prob_{match}(s_i, s_j) = \frac{c}{distance(s_i, s_j)^{\gamma}}$$

where c is a normalizing factor. This distribution simulates the scenario that users have more interest in local events. In our experiments, γ is set to 1.

Since our focus is not on actual matching algorithms, for simplicity, we use integers on [0, 100,000] as publication and subscription values and perform only equality matching.

5.1.3 Simulation input and output

We simulate CBF as in [10], Channelization as in [22], and MEDYM with the three routing algorithms: MST, CloseMST and BalancedMST. In CBF, servers are organized into one minimum spanning tree. In Channelization, we cluster events into 50 event channels using Forgy K-means algorithm. The configuration for CloseMST and BalancedMST are the same as in Section 4. We assume that an event message has content of 200 bytes, TCP/IP header of 44 bytes, and server IDs in the destination lists takes 16 bits each.

We focus on two key parameters of a pub-sub network scenario: the network size, i.e. number of total pub-sub servers, and *matching ratio* of the servers. *Matching ratio* is defined as the probability that an average event matches any subscription on a pub-sub server. It is the product of the average number of users per server, number of subscriptions per user, and percentage of events each subscription matches. In this way, it summarizes the subscription selectivity of a pub-sub server as a whole. Low matching ratio means user subscriptions are highly selective, and vice versa. Unless stated otherwise, we simulate a network of 1000 pub-sub servers.

5.2 Results

We evaluate pub-sub system performance along the following dimensions:

- Storage and management cost, measured by percentage of global subscription information an average server has to store and match with.
- Network efficiency in event routing: this includes total network resource usage and network load carried by pub-sub servers and network links.
- Destination list overhead, in terms of extra bandwidth it consumes in event transmission.
- System reliability, measured by the percentage of events lost in the face of pub-sub server failures.
- Scalability of pub-sub system with increasing number of subscriptions and network size.

5.2.1 Subscription replication

First, we look at cost of subscription replication at pub-sub servers. The three approaches maintain remote subscription information in different ways. Both CBF and MEDYM are able to benefit from subscription aggregation techniques to summarize subscriptions from the same direction or server, but at different levels. In Channelization, servers maintain event space partition result rather than subscription content. The amount of information depends on the granularity of specific partitioning method.

Therefore, it is difficult to directly compare the size of storage used for subscription replication in different schemes. Rather, we measure the percentage of global subscriptions that each pub-sub server needs to know about, or equivalently, percentage of servers at which a subscription is replicated on average. Higher subscription replication ratio indicates not only higher storage cost, but also more network traffic and management overhead for subscription update, and the dependence on synchronous, correct operation of more servers.

Without using advertisements, all three approaches require all subscription information be replicated on all servers. Therefore, we focus on the effectiveness of using advertisements to reduce subscription replication. In MEDYM and Channelization, because remote subscriptions are only used in matching with local publications, servers only need to store subscriptions that overlap with their advertisements. A CBF server, in contrast, may have to store subscriptions that are irrelevant to its publication, if it is on a CBF tree path from the subscription server to any publication server with matching advertisements.

In simulation, we randomly choose a subset of servers to be publication servers. Each publication server randomly selects certain events to advertise on. The three figures in Figure 6 show results for having 1%, 10% or 100% of servers as publication servers. In each figure, x-axis shows average

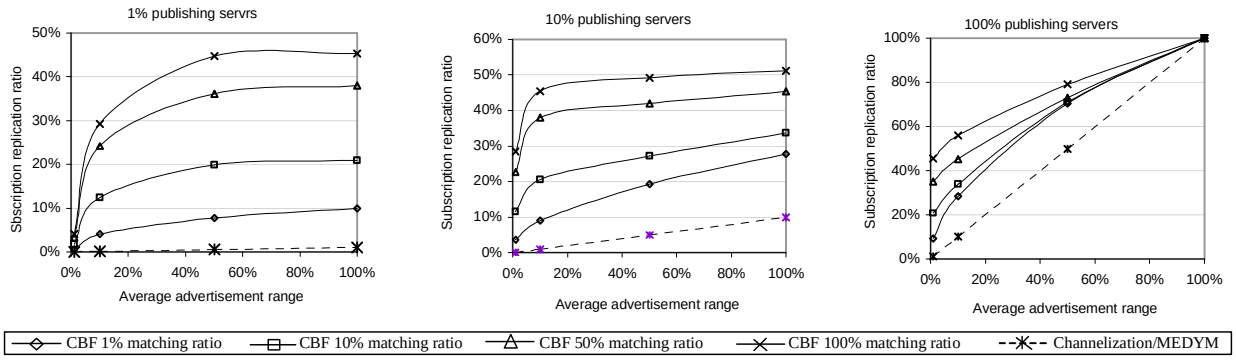


Figure 6. Subscription replication comparison, with uniform publication and subscription distribution.

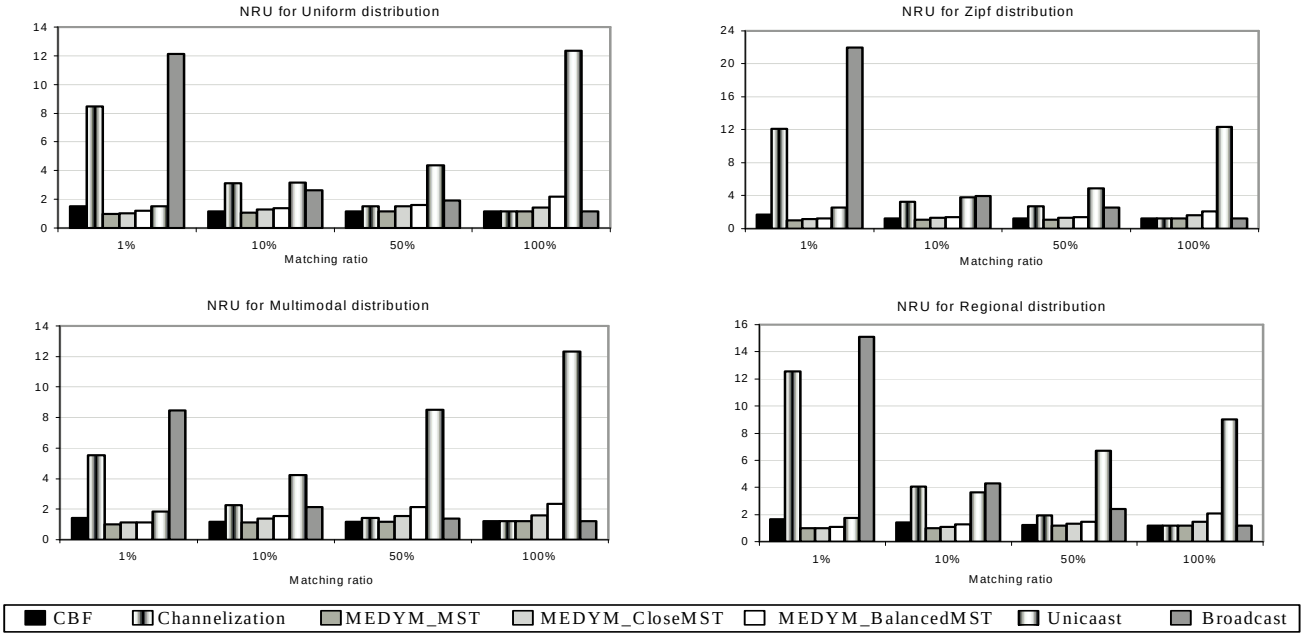


Figure 7. NRU comparison, with various publication and subscription distributions.

Table 2. Server access bandwidth consumption for delivery of one event, with uniform publication and subscription distribution.

Matching ratio	Schemes		Node stress				Link Stress			
			AvgNS	MaxNS	MaxNS/ AvgNS	#Servers traversed	AvgLS	MaxLS	MaxLS/ AvgLS	#Nonzero-stress links
1%	CBF		251	7935	31.6	65	0.05	1.61	30.9	2686
	Channelization		1098	5111	4.65	286	0.26	3.48	13.4	3747
	MEDYM	MST	39	136	3.4	10	0.02	0.25	11.7	3444
		CloseMST	39	136	3.5	10	0.02	0.27	12.1	3452
		BalancedMST	39	60	1.6	10	0.03	0.20	8.1	3464
10%	CBF		1958	15759	14.9	296	0.21	2.65	12.5	2686
	Channelization		3766	21923	5.8	962	0.55	6.96	12.6	2971
	MEDYM	MST	418	2237	5.3	100	0.13	1.48	11.3	3717
		CloseMST	400	1733	4.3	100	0.14	1.34	9.4	3757
		BalancedMST	399	660	1.7	100	0.15	1.21	8.2	3801
100%	CBF		3836	23040	6.0	1000	0.66	6.96	10.5	2485
	Channelization		3836	23040	6.0	1000	0.66	6.96	10.5	2485
	MEDYM	MST	4384	26328	6.0	1000	0.66	6.96	10.5	2485
		CloseMST	4010	20192	5.0	1000	0.54	5.81	10.7	3672
		BalancedMST	4381	14461	3.3	1000	0.64	5.71	8.9	4094

advertisement range, i.e. the average percentage of events a publication server advertises, and y-axis shows subscription replication ratio, i.e. the percentage of global subscriptions an average server stores. For Channelization and MEDYM approaches, subscription replication ratio is always equal to the product of the percentage of publication servers and the servers' advertisement range. For CBF, the result is sensitive to subscription selectivity. Curves in each figure show that with higher matching ratio, a server subscribes to publications from more servers, and its subscriptions have to be stored at more locations. The fraction of extra replication in CBF is most prominent with high publication selectivity, i.e. with small number of publication servers and small advertisement range per server. In such scenarios, most of the subscription replicas are stored on intermediate servers rather than publication servers. These are the situations when advertisement is most needed, but its effectiveness most limited in CBF.

5.2.2 Network efficiency

Next, we evaluate network efficiency per event delivery. The results are average of simulating 10,000 event deliveries.

Normalized Resource Usage (NRU)

We measure the total network resource usage by the summation of latency of network links traversed in event routing. We define NRU as the ratio of the total network resource usage of an event delivery scheme over that of the *ideal multicast*, in which there exists an IP multicast group for delivery of each event.

The results are presented in Figure 7. For comparison, we also simulate two simple event delivery solutions, unicast from publication server to all servers with matching subscriptions, and broadcast to all servers in the system.

Overall, Figure 7 shows that CBF and MEDYM_MST achieve low network resource usage under all circumstances. The approximate MEDYM routing algorithms, MEDYM_CloseMST and MEDYM_BalancedMST, perform less well, as they trade off routing quality for computation efficiency and load balancing. However, when matching ratio is low, even the approximate MEDYM algorithms outperform CBF and Channelization. Results for Channelization have similar trends as broadcast, which indicates that it is not very effective in preventing event from being sent to irrelevant locations. Channelization is also sensitive to data distributions: it performs best with the Multimodal distribution, in which the publication and subscription have consistent distributions with strong locality property; it is less effective with the Uniform distribution, which offers not much clustering opportunity; it performs worst with the Zipf and Regional distribution. This is because with Zipf distribution, most events only match a small number of subscriptions, and therefore the extraneous network traffic ratio is especially high; in Regional distribution, Channelization is especially penalized for sending irrelevant events to far-away servers, while schemes with content-based matching benefit from filtering these events off. When the average matching ratio is high, performance of all the schemes converge to broadcast.

Results from Figure 7 confirm our expectation that per-event content-based matching is critical in achieving high routing accuracy. It also confirms that multicast-like routing efficiency is highly desirable. Unicasting or broadcasting events to individual subscription clients would perform even worse than the inter-server unicast and broadcast schemes examined here; thus, centralized architecture is not expected to be efficient or scalable for a large pub-sub system.

Network load: node stress and link stress

We evaluate network load of a pub-sub system from two aspects: load on pub-sub servers, i.e. *node stress*, and load on network links, i.e. *link stress*. We measure node stress at a node, i.e. a pub-sub server, by the network access bandwidth consumption at that server in event delivery. Note that except for the destination list overhead in MEDYM (which is included in these measurements), the bandwidth consumption are proportional to number of events a server receives. Therefore, the bandwidth results also indicate the overall routing load on pub-sub servers. We measure link stress by number of messages carried by network links in event delivery. The results are shown in Table 2. Due to space constraint, only the results for the uniform distribution are presented in the rest of the paper, as they are the clearest to understand. Results with other data distributions have the same trend as those of uniform distribution and the differences between them are similar to the analysis above.

Network access bandwidth is a precious resource that often directly affects the deployment cost of a pub-sub system. Table 2 shows that MEDYM schemes almost always has lower node stress than CBF and Channelization. The advantage is especially significant when matching ratio is low: to deliver an event with 1% matching ratio, an average CBF server consumes 6.4 times bandwidth than a MEDYM server, and an average Channelization server consumes about 28 times than a MEDYM server. To explain this, we also present average number of servers an event traverses in Table 2. The more servers an event traverses, the higher bandwidth each server spends for routing an average event. Channelization's ability to filter off irrelevant events becomes ineffective quite early: an event that matches only 10% servers is sent to more than 96% servers. CBF is better than Channelization due to its content-based matching ability, but its extraneous bandwidth consumption is still quite high with high subscription selectivity. As in NRU, differences between schemes disappear with higher matching ratio, as all routing methods converge to broadcast. However, MEDYM consumes more bandwidth than the other approaches when matching ratio is 100%. This is due to the overhead of destination lists in MEDYM and will be discussed In more detail later. Interestingly, the higher bandwidth needs due to destination lists is clearly more than outweighed by other considerations in realistic scenarios.

Results for link stress are similar to that of node stress. The number of nonzero-stress network links, i.e. number of

network links that carry event routing traffic, indicates the ability of a routing scheme in utilizing underlying network resources. In CBF, events are only routed through links in the CBF tree. Idle resources on off-tree links may not be used even if these in-tree links become congested. Channelization and MEDYM benefit form higher routing diversity than CBF. Again, BalancedMST is the best in balancing traffic load over network links.

Load balancing

Load balancing is an important feature to achieve high system throughput. Table 2 shows that CBF server load is highly imbalanced. As analyzed in Section 2, servers close to the center of the network carry much higher event traffic than others. For example, in a network of 1000 servers and 1% matching ratio, the server at the center of the CBF tree routes for $1-(1/2)^m = 99.9\%$ events, while a server at the edge of the tree only receives the 1% events that it has interest in. Channelization provides more load balancing opportunity as it distributes event traffic onto different trees. MEDYM schemes generally have well balanced server load due to the high routing diversity. BalancedMST is shown to be especially effective in load balancing. With matching ratio of 1%, the most heavily loaded server in BalancedMST consumes bandwidth that is only 1% of that in CBF and Channelization.

5.2.3 Destination list overhead

Let us examine destination list overhead more closely, since it was an area of potential concern for MEDYM. Figure 8 shows the average destination list size in the above experiments. The curves confirm our expectation that with a random multicast tree topology, the average destination list size is of O(log*m*), where *m* is the number of matching servers. Figure 8 also shows that different versions of MEDYM have different destination list size. This is because the routing algorithms generate multicast trees with different shapes. The MST algorithm generates the most skinny tree, while the CloseMST generates a flatter one, as it considers choosing close servers to be next-hop servers in a more greedy fashion. BalancedMST is largely similar to CloseMST, but it sometimes detours event messages through far away servers for load balancing. The effect is especially apparent with high matching ratio, when servers are likely to be overloaded.

The overhead of destination list adds to bandwidth consumption in MEDYM. Exactly when does MEDYM consume more bandwidth than CBF and Channelization in event delivery? To answer this question, we plot average server bandwidth consumption versus matching ratio in Figure 9. It shows that MEDYM consumes more bandwidth than CBF_MST when matching ratio is higher than 75% and surpasses Channelization when matching ratio is higher than 90%. CBF_CloseMST has lowest bandwidth overhead and surpasses CBF only when matching ratio is higher than 95%. These numbers are dependent on the size of event message payload, and the shape of multicast trees generated in each scheme. In all, destination list overhead is lower for pub-sub systems with low matching ratio and large event messages.

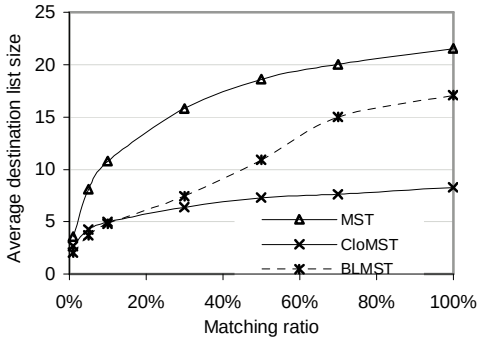


Figure 8. Average destination list size in MEDYM.

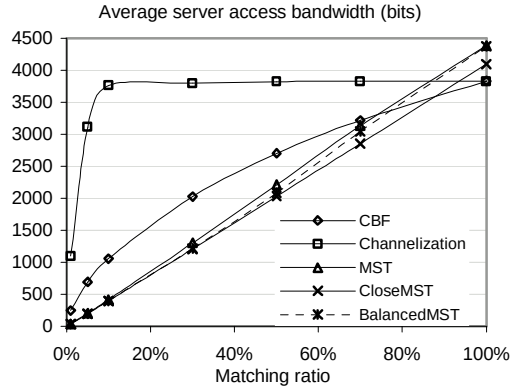


Figure 9. Average server access bandwidth consumption.

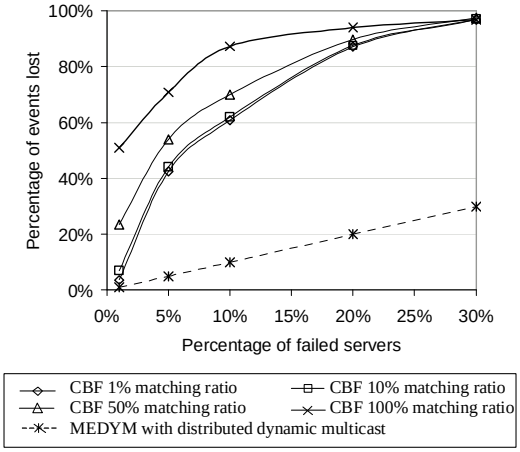


Figure 10. Pub-sub service reliability in the face of server failures.

Interestingly, the point at which MEDYM destination list overhead hurts MEDYM efficiency relative to other systems is close to a point at which it may as well switch to broadcast.

5.2.4 Reliability

Reliability is an important yet challenging issue in distributed pub-sub systems. We compare system reliability in terms of percentage of events lost in the face of server failures.

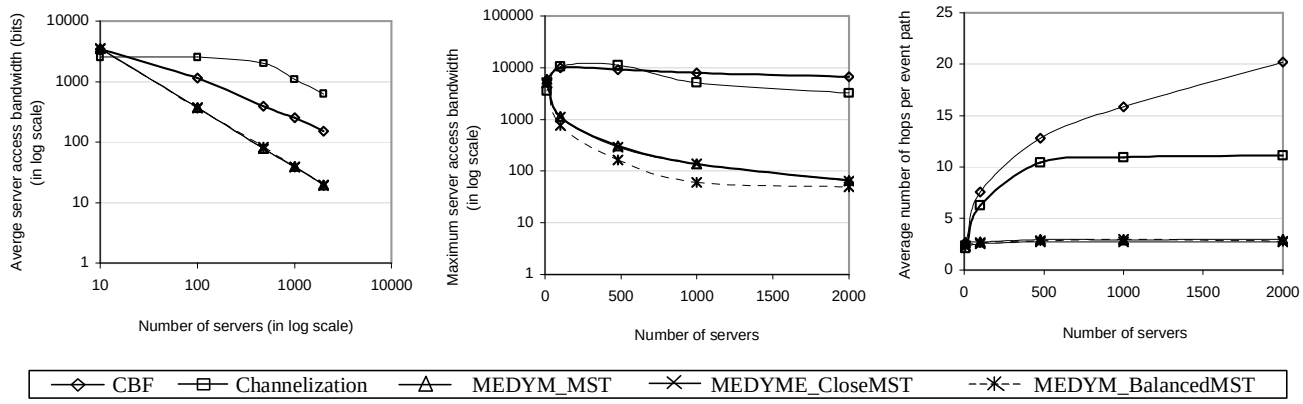


Figure 11. Scalability with increasing network size and fixed number of total subscriptions.

Because failure recovery mechanisms have not been explicitly stated in CBF or Channelization approach, we focus on the *static resilience* of the approaches, i.e. how many events are lost in the face of server failures, before system has recovered. In simulation, we assume that a certain fraction of servers are down, and examine what fraction of the events is lost due to the server failures. In CBF, Channelization, and MEDYM with source-based dynamic multicast, the failed servers silently drop the events they receive. For MEDYM with distributed dynamic multicast, the server that intends to forward the event to a failed server detects the failure (because of the reliable inter-server communication) and rerun the routing algorithm to route events around failed servers. Figure 10 shows the results for CBF and MEDYM. Results for Channelization is very similar to CBF and not shown for clarity. Results for the three MEDYM routing algorithms are also very similar and so only one curve is drawn. In MEDYM, an event arrives at all running servers with subscriptions. In CBF, when a server goes down, the CBF tree becomes disconnected and the events cannot reach the other side of the tree. As shown in Figure 10, in CBF, percentage of lost events increase with number of failed servers quickly. More than half of events are lost with 10% failed servers. It also increases with higher matching ratios, when an average event is routed through more servers and has a higher probability of being dropped.

Note that we only look at the static resilience case here. We show that MEDYM servers can quickly adapt routing decision on environment changes, while the other approaches often have to wait for a system-wide action to be taken.

5.2.5 Scalability

We study how a pub-sub service network scales with two factors: total number of subscriptions and total number of servers in the network. Because we focus on per event delivery efficiency, the total number of publications is not considered.

Three scaling scenarios are listed in Table 3. The analysis above focused on the first scenario. For the second scenario, most of the metrics stay constant because of the constant matching ratio. The results can be inferred from the results above with the same matching ratio and therefore are not shown here. Next, we examine the third scenario. This is the case that shows how increasing number of servers in a pub-sub service network cooperate and share the load of event delivery.

Table 3. Pub-sub service network scaling scenarios.

Scenario	Total #sub	Total #servers	#sub/server	Matching ratio
A	↑	–	↑	↑
B	↑	↑	–	–
C	–	↑	–	↓

Figure 11 shows scalability results for networks with 100 to 2,000 servers. The first figure plots the average server access bandwidth, as an indication of routing load on each server, in log-log scale. Curves for MEDYM schemes are straight-line, indicating that server load are inversely proportional to the network size, which is expected in a well scalable system. However, this is not true for CBF and Channelization. The curve for CBF can be regressed to $y=14732x^{-0.5893}$ with R-square value of 0.9934. The curve for Channelization only starts to decrease with network of more than 500 servers, beyond which matching ratio is low enough for the clustering algorithm in Channelization to be effective. These results indicate that in CBF and Channelization, increasing network size introduces extra routing overhead and the designs do not scale very well in this scenario.

The second figure in Figure 11 shows that the routing load on the most heavily loaded server in CBF and Channelization cannot be relieved with the presence of more servers. In MEDYM, servers only route for events they are interested, and the number of such events decreases with the matching ratio, even for the most heavily loaded server. Especially, the curve for BalanedMST can be regressed to $y = 44011x^{-0.9131}$ with R-square value of 0.9914, indicating well balanced sharing of routing load among servers.

Finally, the third figure in Figure 11 shows the average event path length in terms of number of overlay hops. CBF grows fastest because this number is always equal to the diameter of the CBF tree, which is approximately logarithmic to the network size. Curve for Channelization stays largely flat with more than 500 servers, when the clustering starts to be effective. However, even lower matching ratio cannot further

reduce this number, because the average size of the multicast trees is limited by the fixed number of channels available. MEDYM curves stay flat because the size of the dynamic multicast trees stay constant, as the average number of subscriptions per server is constant.

These results show that MEDYM scales with pub-sub network size more gracefully than the other approaches.

6. IMPLEMANTATION RESULTS

To validate the MEDYM architectural design and test its performance on real network, we deployed a prototype of MEDYM, called Eos, on PlanetLab test bed. We run Eos servers on 56 PlanetLab sites, 44 in United States and 12 abroad. We used two different ways for input of server network location information: pair-wise ping and GNP [21] with 8 dimension Euclidean space. Results such as bandwidth consumption and destination list overhead are consistent with the simulation results above and are not shown here. Figure 12 shows the Relative Delay Penalty (RDP) for event paths in Eos using MST and CloseMST routing algorithm. RDP is defined as the ratio of end-to-end event path latency in Eos system to the IP latency (measured by pair-wise ping) between the publishing and subscription servers. Figure 12 shows that pair-wise ping provides more accurate server location information than GNP. We found that the error from GNP mainly comes from the scenarios where latencies between servers are inconsistent with their network locations: for example, servers that are close in geographic locations often observe similar latencies to the landmark nodes and therefore obtain similar coordinates. However, the real IP latency between them can be quite high, probably due to a congested network link or specific ISP configuration. This can also be seen from Figure 13, which shows that all event paths with high RDP have small IP latencies. Figure 12 and Figure 13 also show that about 10% event paths have RDP less than 1, which indicates that overlay forwarding can take less time than IP forwarding, especially when IP latency between servers are high.

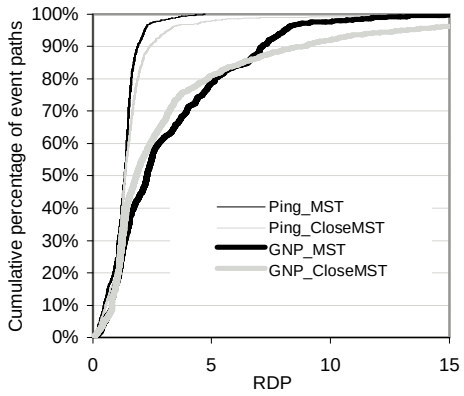


Figure 12. Cumulative Distribution Function for Relative Delay Penalty of event delivery paths on PlanetLab.

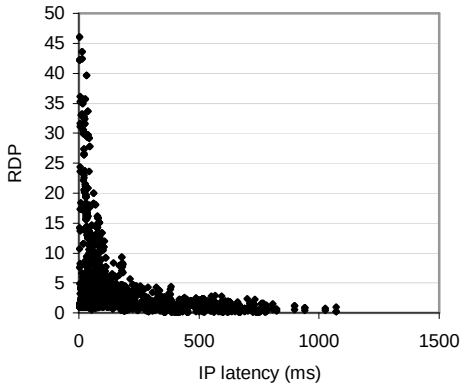


Figure 13. RDP vs. IP latency in GNP_CloseMST experiment. Each point denotes the existence of an event path with a given IP latency and RDP.

7. RELATED WORK

Many distributed pub-sub system designs adopt the content-based forwarding (CBF) approach. In JEDI [13], a hierarchical event routing scheme is proposed, but it was found to perform inferior to the peer-to-peer topology in [9]. In Gryphon [3], a link-matching algorithm is designed to partially match the event with an annotated network topology data structure to determine the directions to forward the event to. [11] designed detailed matching algorithm for content-based forwarding. [8] proposes to build multiple small CBF networks for a pub-sub system, so that event routing in each network can be implemented with lower cost. However, because event routing is still by content-based forwarding, it inherits the tradeoffs of CBF.

The problem of delivering an event message to a group of distributed users is similar to the traditional multicast problem. Application-layer multicast [4][12] is expected to scale better than IP multicast mainly because an end host only routes messages for groups that it participates. This is similar to the idea in MEDYM that servers only route for interesting events. In comparison, dynamic multicast is completely stateless so that no session information needs to be established or managed. Explicit Multicast [5] is another technique of stateless multicast. Dynamic multicast is different from Explicit Multicast mainly in three ways: first, dynamic multicast only route events through destination servers, and the destination list overhead is a function of the number of such servers; Explicit Multicast routes messages through many intermediate IP routers and therefore cannot scale with large destination sets; Second, dynamic multicast paths are dynamically computed for each event message received. Finally, dynamic multicast may be implemented in a distributed way that is highly flexible and resilient.

8. CONCLUSION AND FUTURE WORK

We have examined architectural approaches for distributed, content-based publish-subscribe service networks, and have proposed a new approach called MEDYM (Match Early with DYnamic Multicast). We compared MEDYM with two major existing design approaches: content-based forwarding

(CBF) and Channelization. MEDYM shares the advantage with the Content-based Forwarding approach in that destination servers for a message are determined accurately based on content-based matching against subscriptions. It shares the advantage with Channelization in that after event content is matched once, subsequent event forwarding is through simple, fast address-based routing rather than expensive content-based routing. In this way, MEDYM is a design approach following the end-to-end argument [26], extracting the complex content-based matching functionality out of the routing network, while CBF is more of an active network [27] approach, developing network intelligence.

Unlike the existing approaches, MEDYM does not assume static network topologies for event delivery. Instead, it uses *dynamic multicast* for event routing, which matches the highly diversified communication pattern in pub-sub system with high routing diversity. Extensive simulation shows that dynamic multicast achieves high network efficiency and low operation cost as well as low management overhead. It allows for flexible routing optimization and distributed routing decision-making that helps to improve system reliability. We examined potential overhead introduced in MEDYM, mainly the destination list size and real-time routing computation, and found that they are well manageable and more than over-weighted by the benefits MEDYM brings.

Compared to the existing approaches, advantage of MEDYM is most prominent with large-scale pub-sub service network with high user interest selectivity. We observe that this is exactly the scenario where content-based pub-sub is most attractive, and intelligent and efficient event routing is most needed. Therefore, we believe that MEDYM is a promising architectural design, with the dynamic multicast scheme being potentially applicable in contexts other than pub-sub as well. We plan to further build out, scale and test the real system, and use it to deploy publicly available pub-sub services and hence generate real workloads (a key limitation going forward in pub-sub research).

References:

[1] http://mobile.yahoo.com/wireless/alert

[2] http://www.planet-lab.org

[3] M. K. Aguilera, R. E. Strom, D. C. Sturman, M.Astley, and T. D. Chandra, "Matching events in a content-based subscription system," In *Eighteenth ACM Symposium on Principles of Distributed* Computing, 1999.

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. "Scalable Application Layer Multicast". In *Proc. of ACM SIGCOMM*, 2002.

[5] R. Boivie et al., "Explicit Multicast (Xcast) Basic Specification", draft-ooms-xcast-basic-spec-03.txt

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," In *INFOCOM*, 1999.

[7] L. F. Cabrera, M. B. Jones and M. Theimer, "Herald: Achieving a Global Event Notification Service," In *Proc. of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.

[8] F. Cao, J.P.Singh, "Efficient event routing in content-based publish-subscribe service network". In Proc. *IEEE INFOCOM* 2004.

[9] A. Carzaniga, "Architectures for an Event Notification Service Scalable to Wide-area Networks". PhD Thesis. 1998.

[10] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and evaluation of a wide-area event notification service," In *ACM Trans. on Comp. Sys.*, 2001.

[11] A. Carzaniga, A.L. Wolf, "Forwarding in a Content-Based Network". In *Proceedings of ACM SIGCOMM 2003*.

[12] Y. H. Chu, S. G. Rao and H. Zhang, "A case for end system multicast," in *ACM SIGMETRICS*, 2000.

[13] G. Cugola, E. Di Nitto, A. Fuggetta, "The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS", in *Proc. Of IEEE Transactions on Software Engineering*, 2001.

[14] *D. ELY, S. SAVAGE et al.* "Alpine: A user-level infrastructure for network protocol development." *In* Proc. 3rd USITS *2001.*

[15] P. Francis, S. Jamin, et al, "IDMaps: a global internet host distance estimation service". In Proc. *IEEE/ACM Trans. Netw. 9*(5): 525-540, 2001

[16] Z. Ge, M. Adler, J. Kurose, D. Towsley and Steve Zabele, "Channelization problem in large scale data dissemination," Technical report, University of Massachusetts at Amherst, 2001.

[17] R.Gummadi, S.Gribble et al. "The Impact of DHT Routing Geometry on Resilience and Proximity", In *ACM SIGCOMM* 2003.

[18] E. N. Hanson, C. Carnes, L. Huang, M. Konyala, "Filtering Algorithms and Implementations for Very Fast Publish/Subscribe Systems," In *Proc. of ACM* SIGMOD, pages 115-126, 2001.

[19] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet Topology Generator," Tech. Rep. CSE-TR-433-00, EECS Department, University of Michigan, 2000

[20] A. Nakao, L. Peterson, "A routing underlay for overlay networks", In *Proc. ACM SIGCOMM* 2003.

[21] T. S. E. Ng and H. Zhang. "Predicting Internet Network Distance with Coordinates-Based Approaches." In Proc. *IEEE INFOCOM* 2002.

[22] A. Riabov, Z. Liu, J. Wolf, P. Yu and L. Zhang, "Clustering Algorithms for content-based publication-subscription systems," In *ICDCS* 2002.

[23] A. Riabov, Z. Liu, J. Wolf, P. Yu and L. Zhang, "New Algorithms for content-based publication-subscription systems", In ICDCS 2003.

[24] R. Shah, R. Jain, F. Anjum, "Efficient Dissemination of Personalized Information Using Content-Based Multicast," In *IEEE Infocom*, 2002.

[25] N. Spring, R. Mahajan, et al, "Measuring ISP Topologies with Rocketfuel". In SIGCOMM 2002.

[26] 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.

[27] D. Tennenhouse, J. Smith, et al. "A Survey of Active Network Research". In *IEEE Communications Magazine*, Vol. 35, No. 1, pp80-86. January 1997

[28] T. Wong, R. Katz, and S. McCanne. "An evaluation of preference clustering in largescale multicast applications," In Proc. IEEE INFOCOM 2000.

[29] T. Yan and H. Garcia-Molina. "SIFT---A tool for wide-area information dissemination". In *Proc USENIX TECH CONF*. 1995.