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

Fengyun Cao
Princeton, NJ 08540, USA
fcao@cs.princeton.edu

Jaswinder Pal Singh
Princeton, NJ 08540, USA
jps@cs.princeton.edu

Abstract

Architecture design for content-based publish-subscribe service networks has been a challenging problem, because its communication paradigm cannot be directly supported by existing network primitives. In this paper, we propose a new architectural design called MEDYM & Match Early with DYnamic Multicast. Unlike existing approaches, MEDYM does not rely on static overlay networks for event delivery. Instead, an event is matched against subscriptions early at the publishing server, to identify destinations with matching subscriptions, and then sent to destinations through a dynamically constructed multicast tree. This architecture achieves low computation cost in matching and high network efficiency in routing.

We evaluate MEDYM through detailed simulations, and compare it with the two major existing design approaches: Content-based Forwarding and Channelization. Results show that MEDYM significantly improves event delivery efficiency and system scalability. We also examine closely overheads introduced in MEDYM, and found them to be well acceptable and more than outweighed by the benefits of the approach. We expect the MEDYM architecture to scale to pub-sub networks of thousands of servers, which we believe is adequate for many interesting applications in the foreseeable future.

1 Introduction

Content-based publish-subscribe (pub-sub for short) is an important paradigm of asynchronous communication between distributed entities. In such systems, users subscribe to content-based conditions, and will be notified when other users publish events to the system that satisfy their conditions.

In this paper, we study architecture design for large-scale content-based pub-sub service networks. As shown in Figure 1, in such a network, a set of pub-sub servers are 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. In this paper, we focus on the problem of efficiently delivering events from the servers where they are published to the servers with matching subscriptions. We do not address the communication between pub-sub servers and their clients.

Architecture design for content-based pub-sub networks is challenging for two reasons. First, content-based pub-sub communication is guided by the content of event publications and subscriptions, rather than by network addresses. It is the system's responsibility to match events with subscriptions to identify their destinations. Second, even when the destinations are known, traditional group-based multicast techniques [6] cannot be readily used to route events to all destinations. This is because content-based subscriptions are usually highly diversified, and different events may satisfy the interests of widely varying sets of servers. In the worst case, the number of such sets can be exponential to the network size ($2^n$ where $n$ is the number of servers), and it is impractical to build a multicast group for each such set.

Next, we review major existing design solutions for content-based pub-sub networks and discuss their trade-offs in Section 2. In Section 3, we propose a new design approach called MEDYM: Match-Early with DYnamic Multicast. We analyze its benefits and challenges, and describe in detail efficient implementation techniques. In Section 4 we present simulation evaluation of MEDYM, in comparison with existing approaches. In Section 5, we conclude the paper and identify directions for future work.

![Figure 1. Example of a publish-subscribe service network.](image-url)
2 Related Work

Existing content-based pub-sub network architecture design can be largely categorized into two classes, which we call the Content-based Forwarding (CBF) approach [4][5][7] and the Channelization approach [8][11][14].

2.1 Content-based Forwarding (CBF)

As shown in Figure 2, in a CBF network, pub-sub servers organize a priori into one or more acyclic overlay networks, which we call CBF trees. Every server maintains a forwarding table that records sum of subscriptions from each direction on the tree. A published event is sent along the CBF tree, matched with subscriptions in the forwarding tables at every step, and forwarded only towards directions with matching subscriptions.

The major drawback of the CBF approach is its high computation and maintenance cost. First, the per-step content-based matching is a computationally expensive operation. Second, events are often routed through intermediate servers without matching subscriptions, resulting in extra workload on these servers. Servers and network links close to the center of the network are especially likely to carry such irrelevant event traffic and become system bottlenecks. Finally, when CBF tree topology changes, such as to adapt to dynamic network environment, the forwarding tables also need to change, which may require transfer of large amount of subscription data.

2.2 Channelization

The central idea in the Channelization approach is to utilize existing multicast techniques for event delivery. As shown in Figure 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 that subscribe to any event in that channel. A published event is multicast within its channel to all group members. The process of address-based multicast is usually much simpler and faster than the content-based forwarding in CBF.

The main challenge for Channelization approach is that the number of multicast groups it can build is often much smaller than the number of possible event destination sets (see Section 1). As a result, the same event channel often has to accommodate events with substantially non-overlapping destination sets, and servers can easily receive irrelevant events from groups that they join for other events.

2.3 Observations

Based on the analysis above, we make two observations. First, fine-grained content-based matching of events with subscriptions, albeit computationally expensive, is necessary to curtail extraneous event traffic and achieve high routing accuracy. Second, using static overlay networks, both CBF and Channelization cannot avoid sending events to irrelevant servers, incurring unnecessary processing and network load. These observations motivate us to seek for new, unconventional design approaches.

3 MEDYM Design

We propose a new architecture for content-based pub-sub network called MEDYM: Match Early with Dynamic Multicast. Unlike existing approaches, MEDYM does not build static overlay networks for event delivery. Its event delivery process is as shown in Figure 4. When an event is published, it is first matched against subscriptions from remote servers, to obtain a destination list of servers with matching subscriptions. Then, the event is routed to the destination servers through dynamic multicast: On receiving an event message, based on its destination list, a server dynamically computes the next-hop servers to which to forward the message, as well as the new destination list for each of the next-hop servers. In this way, a transient dynamic multicast tree is constructed on the fly. Along this tree, the event is routed to all servers with matching subscriptions.

MEDYM can be seen as following the End-to-end argument in distributed system design [12]: It decouples
the pub-sub service into two functionalities: complex, application-specific matching at network edge, and simple, generic routing solution in the network. This architecture offers several advantages:

- **Low computation cost.** In MEDYM, an event is matched with subscriptions only once. The rest of event delivery process is through simple, address-based dynamic multicast routing. Compared to CBF approach, this achieves the same high matching accuracy at lower computation cost.

- **Minimum traffic load with proper distribution.** In MEDYM, servers only receive (and route for) the events matching their subscriptions. This not only minimizes total event traffic load on pub-sub servers, but also distributes the load consistently with servers’ self-interests. These two effects result in efficient server resource usage and increased system scalability. Given the heterogeneous server interests in content-based pub-sub networks, they also provide an important incentive for servers to participate in the network. In comparison, neither CBF nor Channelization could achieve this property using static event delivery overlays.

- **High network efficiency.** By customizing multicast routes based on individual event traffic patterns, dynamic multicast allows for fine-grained network efficiency optimization in event routing.

- **Easy deployment and management.** MEDYM servers are loosely coupled by soft states, and do not need to maintain specific overlay topology. With content-independent dynamic multicast as the routing primitive, MEDYM can seamlessly support different pub-sub applications and upgrade to new data types or matching semantics.

In this paper, we treat the relatively well-studied content-based matching problem as an available plug-in module and do not discuss it further. Dynamic multicast is a novel multicast scheme we propose specifically to support the diversified routing need in pub-sub networks. Next, we discuss its efficiency and effectiveness in detail, with an emphasis on the overheads it introduces, such as real-time route computation and traffic overhead of destination lists.

As shown in Figure 5, dynamic multicast 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, a routing algorithm $f_i$ runs as follows:

$$<n_i, DL_i> = f_i(DL) \quad i = 1 \cdots d$$

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

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

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

(c) $n_i \in DL_i$

These invariants guarantee that the message is sent to all its destination servers, and to each server at most once. Routing loops and redundant paths are prevented. The whole multicast tree is thus resolved in a recursive way.

![Figure 5. Dynamic multicast routing.](image)

**3.1.1 Distributed Dynamic Multicast**

To avoid the fragility of centralized decision-making, we propose distributed dynamic multicast routing, in which each server independently computes its local part of the multicast tree $f_i$ its next-hop servers. A server 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 benefits from the fact that servers in a distributed network often have more accurate and/or up-to-date knowledge about local environment than about distant areas. It is also highly resilient to failures. When a server fails to deliver a message to next-hop server $n_i$, it simply re-runs $f_i(DL, n_{j})$ so that the message can bypass $n_i$ and still be delivered to other servers in $DL_i$. 

**3.1.2 Routing Algorithms**

In this paper, we use network link latency to loosely measure the data transmission cost on a network link. To minimize total network cost, we first experimented with routing algorithm that constructs 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 traffic throughput.
We then developed algorithm SPMST, for Short-Path-MST, as shown in Figure 6. It computes an approximate minimum spanning tree across destinations in a fast and distributed way. Offline, an array called ShadowBitVectors is maintained to help quickly identify next-hop servers. We say that server \( s_j \) is shadowed by server \( s_i \), if \( s_i \) is closer to \( s_j \) than to current server \( s_k \). Under this condition, \( s_j \) would forward the message to \( s_k \) at lower (latency) cost than \( s_j \) 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.

```java
computeShadowBitVectors(s) { // offline
    foreach server \( s_i \) {
        foreach server \( s_j \) if (\(<\text{DistanceMatrix}[i][j]\) \&\& \(\text{DistanceMatrix}[i][s] \&\& \text{DistanceMatrix}[s][j]\])
            Set_\text{th} \text{bit in ShadowBitVector[}\text{]} = 1
    }
}

SPMST(DL) { // online
    Nexthops = DL;
    foreach server \( s_i \) in DL
        if (ShadowBitVector[i] & DLBitVector \(!= 0\))
            Nexthops_remove(s_i);
    if (\(\text{Nexthops} = \text{maxNexthops}\))
        Nexthops = closest_nexthops(maxNexthops);
    foreach server \( s_j \) in (DL-Nexthops) {
        \( n_j = \text{closest_nexthop}_\text{to}(s_j) ; \)
        DL_i += \[s_j] ;
        return(\(n_j, DL_n) ;\)
    }
}
```

Figure 6. SPMST routing algorithm.

Table 1 shows the computation time of MST and SPMST algorithm for certain number of destinations. 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 high routing throughput. Furthermore, note that the average destination list sizes in dynamic multicast messages, as analyzed in Section 3.1.4, are much shorter than the DL sizes in the table. In comparison, studies ([1][5]) show that it typically takes hundreds of milliseconds to match an event with large number of content-based subscriptions. This confirms our expectation that address-based multicast forwarding in MEDYM is much simpler and faster than content-based forwarding in CBF.

### Table 1. Computation time of dynamic multicast routing algorithms, with destination list size [DL].

<table>
<thead>
<tr>
<th>Routing algorithm</th>
<th>Computation time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[DL]=100</td>
</tr>
<tr>
<td>MST</td>
<td>1.8</td>
</tr>
<tr>
<td>SPMST</td>
<td>0.08</td>
</tr>
</tbody>
</table>

#### 3.1.3 Caching Routing Decisions

An interesting question is whether there is enough temporal locality among dynamic multicast destinations so that the routing decisions can be effectively cached. Because caching effectiveness highly depends on application data distribution, we plan to examine it in the context of real pub-sub workload, and do not assume caching as a general solution here.

#### 3.1.4 Destination List Size

Destination lists in dynamic multicast message headers introduce network traffic overhead. However, as a message’s destination list size is reduced at every step, it is easy to prove that the average size over all edges in a multicast tree is about equal to the diameter of the tree. This is confirmed by Figure 7, which shows that as the diameters of the multicast trees, computed by SPMST algorithm, 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, on average, each copy of the message carries only 8 server IDs in its destination list. Such overhead is quite acceptable, especially considering that event messages often carry rich content, such as attribute-value pairs or full-text documents.

However, distribution of destination lists may not be well balanced, as the lists are longer at locations close to the publisher. We explore this issue further with evaluation results for different publisher distribution scenarios in Section 4.

![Figure 7. Dynamic multicast tree diameters and average destination list sizes.](image)

#### 3.2 State Maintenance

A MEDYM server maintains two kinds of information about other servers in the system: the sum of their subscriptions, used in match-early phase, and their network locations, used for computing network-efficient multicast routes. Due to space limitation, here we only state the fact that such requirement of global knowledge does not introduce much overhead in MEDYM, compared with existing approaches. For example, to effectively quench ([13]) publication of garbage events that match no subscription, publishing servers must know at least the sum of all subscriptions in the system. We have also shown that due to its routing flexibility and robustness, MEDYM is able to benefit from state-of-the-
art network location estimation techniques [10], to obtain approximate (yet accurate enough for the purpose) server location information with little overhead. Interested readers can refer to [3] for further analysis.

4 Simulation Evaluation

We developed a message-level, event-driven simulator for evaluation. We simulate a pub-sub network of 1000 servers, randomly attached to 2500 IP routers. The IP network is generated using the transit-stub model of GT-ITM Internet topology generator [2]. An event message has payload of 200 bytes and TCP/IP header of 44 bytes. 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 simulate MEDYM using the SPMST algorithm described above. For existing approaches, we simulate two versions of CBF: CBF_MST as in [4], where a single CBF tree is built as minimum spanning tree across all servers, and CBF_SPT as in [5], where CBF trees are shortest path trees rooted at publishing servers. We simulate Channelization approach as in [11], using Forgy K-means algorithm to cluster events into 50 event channels. The algorithm was found to generate the best partition results in the paper.

To evaluate event delivery efficiency of different approaches, we look at work load generated on pub-sub servers and network links under various subscription scenarios. We define matching ratio to be the average percentage of events a server matches. Low matching ratio means user subscriptions are highly selective and diversified, and vice versa.

4.1 Server processing load

To avoid dependence on specific matching algorithm performance, we measure event processing load on a server by the amount of events it receives in the process of event delivery. Figure 8 plots this number, normalized as the percentage of total number of events published. Figure 8 shows that Channelization servers receive the most amounts of events, indicating its ineffectiveness in filtering out extraneous event traffic. CBF servers receive fewer events, due to its accurate per-step filtering. MEDYM servers receive the least possible amount of events, i.e. only the events that they subscribe to. 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% in MEDYM, 8% events in CBF_MST, 9% in CBF_SPT, and 29% in Channelization.

Figure 9 presents the results under the traditional metric goodput, defined as percentage of events a server receives that match the server’s interest. In both CBF and Channelization, server goodput degrades with low matching ratio, while MEDYM always achieves 100% goodput.

Next, we look at load distribution on pub-sub servers. Figure 10 shows the cumulative percentage of servers that receive no more than certain number of events, when each server subscribes to 10% of total events. At the left end, all MEDYM servers receive 10% events each; at the right end, most Channelization servers receive more than 90% of total events. In the middle, CBF servers bear highly imbalanced load: about 40% servers receive only 10% events each, while 20% servers in CBF_MST and 10% in CBF_SPT receive more than 80% events each. The reason is as analysis in Section 0: servers at the center of the CBF tree(s) route for many more (irrelevant) events than their peripheral peers, and are likely to become bottlenecks for system throughput and scalability.

4.2 Server bandwidth consumption

Figure 11 shows the average bandwidth a server consumes in the processing of delivering one event. Different from Figure 8, MEDYM servers only achieve close to minimum average bandwidth consumption. The difference between the MEDYM curve and the minimum line shows the destination list overhead, which makes MEDYM surpass CBF and Channelization by a small amount when matching ratio is above 90%.

Maximum server bandwidth consumption can be sensitive to publisher distribution, and we study two different scenarios: Figure 12 shows the all-publisher scenario, in which every server publishes the same number of events, while Figure 13 shows the single-publisher scenario, in which only one server (presumably a centralized information publisher) publishes all the events. Under both scenarios, the CBF approaches perform much worse than for the average case. In CBF_SPT, the overlay minimum spanning trees tend to degenerate into star-shape topology with the publishing server at the center, with event routing degrading into unicast. Despite its low routing accuracy, Channelization outperforms CBF under low matching ratios. This is because when event space partitioning is effective, even the heaviest loaded Channelization servers join only a subset of multicast groups, while CBF servers always participate in all CBF trees. Again, we see that destination list overhead does not prevent MEDYM from significantly outperforming the others for most of the cases, showing well balanced network load across MEDYM servers. However, MEDYM performs less well for the single-publisher case, especially when matching ratio is high. Therefore, we developed another dynamic multicast routing algorithm to effectively balance server load, as described in [3].
Figure 8. Average # events received, shown as percentage of all events published.

Figure 9. Average server goodput (#events interested / #events received).

Figure 10. CDF of #events received, with 10% matching ratio.

Figure 11. Average server bandwidth consumption.

Figure 12. Max bandwidth consumption for all-publisher scenario.

Figure 13. Max bandwidth consumption for single-publisher scenario.

Figure 14. Average network link stress.

Figure 15. Max link stress for all-publisher scenario.

Figure 16. Max link stress for single-publisher scenario.

Figure 17. Average event path length, in terms of overlay hops.

Figure 18. Relative Average Delay (RAD) of event paths.

Figure 19. Relative Maximum Delay (RMD) of event paths.
4.3 Network link stress

Next, we look at the traffic load imposed 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 Figure 14, and maximum link stress under the all-publisher and single-publisher scenarios are shown in Figure 15 and Figure 16. These results exhibit similar trends as server bandwidth consumption. In Figure 14 and Figure 15, the differences between different architectures are smaller than in Figure 11 and Figure 12. This is because the transit-stub IP topology we used offers low routing diversity: there are a few long inter-domain network links traversed by all messages and they contribute a large part of the stress. We expect that in larger IP networks the difference between the approaches would be more significant. Figure 16 shows that the load imbalance problem in CBF SPT is especially serious for single-publisher case, where links close to the publishing server are heavily loaded by many copies of the same event message.

4.4 Event delivery latency

In real-time pub-sub applications, it is desirable that events be sent to subscribers with short latency. Event delivery latency consists of the processing latency at intermediate servers and network latency on links.

Figure 17 shows average event path length in terms of overlay hops. In CBF, the event path lengths always equal to the diameters of the (fixed) CBF tree(s). In Channelization, events with few matches may be routed through a small multicast tree and traverse less servers on the path. In MEDYM, the average path length is only about logarithm to the number of matched servers.

For network latency, we define Relative Delay Penalty (RDP) as the ratio of end-to-end event delivery latency in the pub-sub network to the latency of underlying IP routing. Figure 18 and Figure 19 present the average and maximum RDP for all event paths. Using shortest-path trees, CBF SPT always achieves lowest RDP, only slightly higher than 1. MEDYM achieves quite low RDP of about 1.5.

5 Conclusion and future work

In this paper, we presented design and evaluation of a new architectural design for content-based pub-sub service networks, called MEDYM & Match Early with DYnamic Multicast. Simulation results show that MEDYM significantly improves event delivery efficiency, both in terms of server load and network efficiency, compared to existing design approaches. MEDYM is most advantageous when subscriptions are highly selective and diversified, exactly the scenario when intelligent and efficient pub-sub service is most needed. The overheads introduced in MEDYM are acceptable and more than outweighed by the benefits it brings. We also expect MEDYM network to be relatively easy to implement and maintain, compared to pub-sub networks that build static event routing overlay networks.

As part of MEDYM, we designed dynamic multicast as a generic multicast scheme for efficient message delivery to dynamic sets of destinations. We expect its efficiency, flexibility and robustness be valuable for contexts other than pub-sub as well.

We have implemented a prototype of MEDYM on the PlanetLab test bed [9]. Preliminary results have validated our simulation findings and can be found in [3]. We plan to deploy a publicly available pub-sub service network using MEDYM and collect real pub-sub workload for further research.

6 References