Scalable Content-based Publish/Subscribe

Shuping Ji\textsuperscript{1}, Chunyang Ye\textsuperscript{2,3}, Jun Wei\textsuperscript{1} and Hans-Arno Jacobsen\textsuperscript{3}

\textsuperscript{1}Chinese Academy of Sciences, Beijing, China  
\textsuperscript{2}Hainan University, Hainan, China  
\textsuperscript{3}Middleware Systems Research Group, University of Toronto

Working Technical Report, initial version: August 2011

Abstract—Despite suffering from inefficiency and flexibility limitations, the filter-based routing (FBR) algorithm is widely used in content-based publish/subscribe (pub/sub) systems. To address these limitations, we propose a dynamic destination-based routing algorithm called D-DBR, which decomposes pub/sub into two independent parts: Content-based matching and destination-based multicasting. D-DBR exhibits low event matching cost and high efficiency, flexibility, and robustness for event routing for overlays with up to hundreds of brokers. We further extend D-DBR to a novel routing algorithm called MERC for overlays with larger number of brokers. MERC divides the overlay into interconnected clusters and applies FBR and D-DBR to route events inter- and intra-cluster, respectively. We implemented all algorithms in the PADRES pub/sub system. Experimental results show that our algorithms outperform FBR in terms of improving event dissemination throughput by up to 700\% and reducing the end-to-end latency by up to 55\%.

I. INTRODUCTION

Due to its asynchronous nature and inherent decoupling properties, the distributed content-based publish/subscribe paradigm (pub/sub, for short) has been widely used in the design of many distributed applications, such as online news dissemination [21], workflow management [13], business process execution and monitoring [20], multi-player online gaming [3], network management and monitoring [25], mobile alert processing [17], and distributed system monitoring [33], to only name a few.

In the distributed pub/sub paradigm, a number of brokers are interconnected as an overlay network to provide an event notification service, which routes events from a publisher to interested subscribers.

The routing algorithm employed by a pub/sub system is crucial to managing performance, load distribution, and scalability in the system. It is a challenging undertaking to design efficient and scalable routing algorithms for pub/sub for two main reasons: First, an event does not carry any destination information. Instead, brokers identify interested subscribers for every event by matching the event against interest specifications in the form of subscriptions. Second, subscribers’ interests are often highly diversified. The number of potential destination sets is usually very large. Thus, it is difficult to efficiently route each event to its destinations.

The design of efficient event routing, especially, in content-based pub/sub systems is still an active area of research [22]. We observe that even the most widely used filter-based routing algorithm (FBR)\textsuperscript{1} [6], [7], [8], [13], [18] suffers from the following four limitations: (1) Difficulty in supporting general overlay topologies, (2) subscription duplication, (3) redundant and repeated event matching, and (4) lack of flexibility in supporting overlay reconfiguration.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig1.png}
\caption{Limitations of the FBR algorithm}
\end{figure}

\textbf{Difficulty in Supporting General Overlay Topologies:} The original FBR algorithm was designed only for acyclic overlay networks [22]. This design choice makes it difficult to support fault tolerance, load balancing and overlay reconfiguration. Although, Li et al. extended FBR to support general overlays [19], their solution, however, introduces two additional sources of overhead. First, advertisement broadcasting in a general overlay generates a large number of redundant messages, which need to be detected and eliminated. Second, a single subscription needs to be delivered to the same broker more than once if the broker publishes more than one advertisement intersecting that subscription.

\textbf{Subscription Duplication:} In FBR, a subscription is stored at every broker along the routing paths from the source broker of that subscription to brokers with matched advertisements. For example, in Fig. 1(a), subscriptions \(S_1\) and \(S_2\) are redundantly stored at brokers \(B_1, B_2, B_3, B_4,\) and \(B_7\). In many applications, the number of subscriptions is very large: In financial applications, there can be thousands of subscriptions [29]; security and surveillance applications may have millions of subscriptions [10]. In these scenarios, resources can be wasted for storing the duplicated subscriptions.

\textbf{Redundant and Repeated Event Matching:} The FBR algorithm repeats the event matching operation at every intermediate broker along the paths from the publisher to interested subscribers. In the example in Fig. 1(a), an event issued at \(B_1\) is repeatedly matched against all subscriptions stored at

\textsuperscript{1}In this paper, we assume the advertisement mechanism is adopted in FBR, since it often improves the algorithm’s scalability [23].
Our experimental results show that our algorithms outperform PADRES, an open-source content-based pub/sub system [28].

A lack of flexibility in supporting overlay reconfiguration: As shown in Fig. 1(a), assume the load between brokers $B_1$ and $B_2$ is heavy. If the overlay is adjusted to that in Fig. 1(b), the routing efficiency could be improved. However, it is difficult to reconfigure the overlay in FBR because the routing state is stored at all intermediate brokers. Yoon et al. proposed three primitive operations to address this issue [34], but some challenging problems such as how to generate the right sequence of operations are left unsolved.

To overcome the aforementioned limitations, we first propose a novel dynamic destination-based routing algorithm called D-DBR. In this algorithm, the pub/sub system is decoupled into two independent layers: Content-based matching and destination-based multicasting. The matching layer is responsible for event matching, whereas the multicasting layer is responsible for event routing. When an event is issued, it is first matched against subscriptions to identify brokers hosting interested subscribers at the matching layer. Then, the event is annotated with the addresses of those brokers and delivered to them via the multicasting layer. In D-DBR, subscriptions are not stored at any intermediate broker. An event only needs to be matched at its source and destination brokers. In addition, supporting general overlay and dynamic overlay reconfiguration for fault-tolerance and performance optimization becomes possible, as we will demonstrate later.

D-DBR requires extra overhead for the destination address list associated with each event. If the destination address list is long, more bandwidth is used. However, our extensive evaluations show that the resulting overhead is small, especially for small- and medium-sized overlays. For example, in an overlay with 100 brokers, when an event is delivered to 50 destination brokers, on average, each of the generated messages carries only 2.84 destination addresses (the average connection degree of the brokers is 3).

Though D-DBR is an effective solution, some factors limit its scalability: First, each broker needs to know all other brokers in the system. The topology maintenance cost can be expensive in large-scale networks. Second, although the average destination address list size is small, when it is near the source broker of events, the destination list can become rather long. As a result, if some broker issues lots of events, its bandwidth can be impacted by the extra destination list overhead.

To achieve better scalability, we propose a further novel routing scheme called MERC — Match at Edge and Route intra-Cluster that aims to alleviate the shortcomings identified above. MERC divides the overlay into interconnected clusters, and applies FBR and D-DBR for inter- and intra-cluster event routing, respectively. In MERC, each broker needs to only be aware of brokers in the clusters it belongs to. Therefore, the destination list overhead is mitigated. Furthermore, the impact of changes in one cluster can be isolated from brokers in other clusters.

We implemented both algorithms, D-DBR and MERC, in PADRES, an open-source content-based pub/sub system [28]. Our experimental results show that our algorithms outperform FBR in terms of improving the throughput by up to 700% and reducing the communication latency by up to 55%.

In summary, we make the following contributions in this paper: (1) We propose the D-DBR algorithm, which supports dynamic overlay reconfiguration and overcomes the aforementioned limitations of the FBR algorithm (cf. Section III); (2) we propose the MERC routing scheme which hierarchically combines FBR and D-DBR to enable a higher degree of scalability (cf. Section IV); and (3) we offer an extensive evaluation of all algorithms based on both real-world experiments and detailed simulations (cf. Section V).

II. Related Work

We categorize existing event routing algorithms for content-based pub/sub systems into four classes: Event flooding (EF) [24], [12], multicast-based routing (MBR) [27], [30], [31], filter-based routing (FBR) [6], [13], [18], and destination-based routing (DBR) [2], [4]. These solutions balance the tradeoff between routing accuracy, subscription duplication, redundant matching, overlay reconfigurability, and scalability differently.

Event Flooding: In EF algorithm, each event originating at a publisher is flooded to all brokers and then matched against local subscriptions at every broker. This routing mechanism is stateless, and can adapt smoothly to broker overlay changes. It also has low event matching overhead and saves memory resources by storing local subscriptions only at each broker. However, it suffers from the shortcoming that events may be propagated to a large number of uninterested brokers. In a general overlay with cycles, a large number of redundant messages can also be generated. Therefore, this routing mechanism suffers from the issue of low routing accuracy.

The gossip-based pub/sub approach [11] is in this category. It is an extension of the EF algorithm and uses a gossip-style mechanism to distribute events. This approach eases the problem of event duplication. However, events are not guaranteed to be delivered to all interested subscribers. Moreover, events need to be cached and lots of memory resources are required.

Multicast-based Event Routing: In MBR, the event space is partitioned into a large number of disjoint multicast groups, for each of which a multicast tree is built. When an event is issued, it is first mapped to an appropriate group and then multicasted on the corresponding spanning tree. For example, in Fig. 2, groups $g_0$ and $g_1$ are pre-computed. When an event is issued at Broker A, it may be first mapped to group $g_0$, and then multicast on $g_0$’s spanning tree.

MBR improves the routing accuracy. In addition, existing multicast techniques, such as IP multicast [14], [27] and application-level multicast [15] can be used to support event
propagation. However, this solution may need a large number of groups, which is exponential in the number of brokers in the worst case scenario. To trade off the overhead, usually only a limited number of multicast groups are constructed [1]. In this case, subscribers with different interests may be clustered into the same group, and events may be forwarded to some uninterested brokers. For example, in Fig. 2, the issued event is forwarded to all brokers in the spanning tree of group g0, among which, D, F, and H are uninterested brokers.

To improve the routing accuracy and reduce the bandwidth overhead, MBR can apply intelligent clustering algorithms [31], [32] to identify multicast groups. However, the effectiveness of the clustering heavily depends on the locality property of events and subscriptions in applications. As a result, this approach is not applicable to the scenario where subscriptions are highly diversified or the application workload changes over time.

**Filter-based Routing:** In FBR, advertisements are first broadcast, and then the matched subscriptions are delivered to the source brokers of the advertisements reversely along the paths on which the advertisements are broadcast. After that, routing trees rooted at the origin of each publisher are constructed. When an event is issued, routing decisions are made via successive content-based filtering at all brokers from the source to the destinations: Every broker along the way matches the event with its stored subscriptions, and then forwards it only toward directions that lead to matched subscriptions.

In FBR, every broker maintains a subscription routing table (SRT) and a publication routing table (PRT) for subscription and event delivery respectively. An advertisement is stored in the SRT as a \{adv, lasthop\} tuple. lasthop indicates from which broker or client this advertisement comes. A subscription is stored in the PRT as a \{sub, lasthop\} tuple. lasthop also indicates from which broker or client this subscription comes.

This approach has several advantages: First, it achieves better routing accuracy than MBR since events are only forwarded toward interested brokers. In the example presented in Fig. 3, the issued event is not forwarded to Broker D or H. Second, it scales well because each broker only needs to know its neighboring brokers. Finally, it is applicable to applications with highly diversified subscriptions. These advantages make FBR the most widely used approach despite the varied limitations discussed in Section I.

**Destination-based Routing:** In content-based pub/sub systems, the flow of events from publishers to subscribers is driven by the content of events. An alternative is to deliver events between brokers directly based on destination address information. We refer to the approaches that route events based on their ultimate destination addresses as “destination-based event routing”. To the best of our knowledge, there are only a few approaches that explored in this direction, i.e., the “link matching” algorithm [2], the MEDYM algorithm [4], and the DRP algorithm [5].

In the link matching algorithm, each broker has a copy of all present subscriptions, which are organized into a special data structure called parallel search tree (PST). Each broker calculates the subset of links to which an event should be forwarded. This algorithm achieves better matching performance but doesn’t overcome the other limitations of FBR. In addition, the maintenance of PST introduces additional overhead and it requires that subscriptions should not be changed frequently.

Fig. 4 illustrates an example of event routing in MEDYM. In this algorithm, every event is attached with destination addresses and routed based on stateless dynamic multicasting. This approach exhibits some appealing characteristics: Every event is only matched twice and is not delivered through any uninterested broker. However, it still suffers from three limitations: (1) Every broker needs to maintain connections with all other brokers; (2) the routing paths need to be dynamically computed for each event; (3) subscriptions are duplicated at each broker.

The DRP algorithm also adopts the idea of attaching every event with destination addresses. But the destination addresses are stored in a fixed-size bit vector. As a result, DRP suffers from severe scalability limitation.

### III. D-DBR Design

This section presents our D-DBR algorithm, an efficient destination-based routing algorithm. It achieves better routing accuracy than EF and MBR, and overcomes the four limitations of FBR presented above. In addition, in D-DBR, the overlay is partially connected and can be dynamically reconfigured. Events are routed on the pre-computed optimal shortest paths. The advertisement mechanism is adopted to optimize subscription propagation. As a result, D-DBR also overcomes the mentioned shortcomings of MEDYM.

As shown in Fig. 5, in D-DBR, the pub/sub system is decoupled into two independent layers: Content-based matching and destination-based multicasting. The matching layer is responsible for event matching, whereas the multicasting layer is responsible for event routing. When a publisher issues an event at a broker, the event is matched against subscriptions managed by the broker’s matching engine to obtain addresses of interested brokers. The addresses are attached to the event. Then, the event is delivered to the interested brokers by the
multicasting layer based on the event’s destination addresses. Upon receiving an event, a destination broker matches the event against its local subscriptions and directly delivers it to the interested subscribers.

![Fig. 6: Event routing in D-DBR](image)

### A. Content-based Matching Layer

**Routing Tables**: To reduce event matching cost, routing information (advertisements and subscriptions) from local clients and other brokers are stored separately. As a result, each broker maintains four routing tables: Local Subscription Routing Table (L-SRT), Remote Subscription Routing Table (R-SRT), Local Publication Routing Table (L-PRT), and Remote Publication Routing Table (R-PRT). These tables’ internal structure resembles the structure of SRT and PRT in FBR, except that the last hop information is replaced by source hop information.

Let us use the example in Fig. 6 to illustrate these tables. In this example, a publisher, pub-c1, issues an advertisement A1 at Broker A and issues three subscriptions, sub-c1, sub-c2, and sub-c3, each issues a subscription at brokers B, E, and G, respectively. Assume that all subscriptions match the advertisement. The advertisement A1 is stored in the L-SRT of Broker A and in the R-SRT of all other brokers. The subscriptions s1, s2, and s3 are stored in the R-PRT of Broker A and in the L-PRT of brokers B, E, and G, respectively. In L-SRT and L-PRT, the sourceID indicates from which client the advertisement or subscription originated, whereas in R-SRT and R-PRT, the sourceID indicates from which broker the advertisement or subscription originated.

**Message Processing**: In D-DBR, advertisements, subscriptions, and events are delivered as messages. In each message, the sourceID field indicates from which client the advertisement or subscription originated, whereas in R-SRT and R-PRT, the sourceID indicates from which broker the advertisement or subscription originated.

Let us use the example in Fig. 6 to illustrate these tables. In this example, a publisher, pub-c1, issues an advertisement A1 at Broker A and issues three subscriptions, sub-c1, sub-c2, and sub-c3, each issues a subscription at brokers B, E, and G, respectively. Assume that all subscriptions match the advertisement. The advertisement A1 is stored in the L-SRT of Broker A and in the R-SRT of all other brokers. The subscriptions s1, s2, and s3 are stored in the R-PRT of Broker A and in the L-PRT of brokers B, E, and G, respectively. In L-SRT and L-PRT, the sourceID indicates from which client the advertisement or subscription originated, whereas in R-SRT and R-PRT, the sourceID indicates from which broker the advertisement or subscription originated.

**Message Processing**: In D-DBR, advertisements, subscriptions, and events are delivered as messages. In each message, the sourceID field indicates from which client the advertisement or subscription originated, whereas in R-SRT and R-PRT, the sourceID indicates from which broker the advertisement or subscription originated.

The input to Alg. 7 is a message m from the match message queue and the resulting output messages are added to the output queue, as shown in Fig. 5. Messages from local clients and other brokers are processed differently. Lines 1~14 process messages from local clients: (1) An advertisement is first inserted into L-SRT and then attached to the output queue with the IDs of all other brokers as its destinations. (2) A subscription is first inserted into the L-PRT and then matched against advertisements in R-SRT to get the IDs of interested brokers as its destinations. (3) An event is matched against subscriptions in both L-PRT and R-PRT to get the IDs of local interested clients and remote interested brokers as its destinations. In this way, advertisements can be broadcast to all other brokers, subscriptions can be delivered to all other brokers whose advertisements match the subscriptions, and events can be delivered to both local interested clients and remote interested brokers.

**Algorithm 1**: Input: message m from queue match

Output: messages inserted into queue output

1: if m is from a local client then
2:   L-SRT.insert(m)
3:   m.destIDList ← all other brokers’ IDs
4:   else if m is a subscription then
5:     L-PRT.insert(m)
6:     m.destIDList ← other brokers’ IDs
7:   else if m is an event then
8:     L-PRT.insert(m)
9:     m.destIDList ← other brokers’ IDs
10: else
11:    m.sourceID
12:    L-SRT.insert(m)
13:    R-SRT.insert(m)
14:    m.destIDList ← other brokers’ IDs
15:   end
16: if m is a subscription then
17:   R-PRT.insert(m)
18:   m.sourceID
19:   R-PRT.insert(m)
20: else if m is an event then
21:   R-PRT.insert(m)
22:   m.sourceID
23:   R-PRT.insert(m)
24: else if m is an event then
25:   m.sourceID
26:   R-PRT.insert(m)
27: end

Fig. 7: Message Processing in the Matching Engine

Lines 16~26 process messages from other brokers: (1) An advertisement is first inserted into R-SRT and then matched against subscriptions in L-PRT. The matched subscriptions are inserted into the output queue with that advertisement’s sourceID as their destination. (2) A subscription is simply inserted into R-PRT. (3) An event is matched against subscriptions in L-PRT only to get the IDs of local interested clients as its destinations. In this way, when an advertisement is delivered to a broker, the broker’s local matched subscriptions are transferred to the source broker of that advertisement. When an event is delivered to a broker, that event is directly transferred to the broker’s local interested clients.

Separating the routing information from local clients and that from other brokers reduces the message matching cost for the following reasons: (1) Subscriptions from local clients only need to be matched against advertisements from other brokers, (2) advertisements from other brokers only need to be matched against subscriptions from local clients, and (3) events from other brokers only need to be matched against subscriptions from local clients.

### B. Destination-based Multicasting Layer

**Routing Tables**: To route a message to its destinations, the multicasting layer also maintains some routing tables: The Topology Routing Table (TRT) and the Shortest Path Routing Table (SPRT). TRT is a matrix, where TRT(i, j) represents the communication cost between broker i and j. If there is a direct connection between broker i and j, TRT(i, j) = 1, otherwise, ∞. SPRT includes a list of records (destBrokerID, nextBrokerID). Each record indicates the next hop on the optimal path to a specific destination. All brokers share the
same TRT, but each broker has its own SPRT. SPRT is computed from TRT using the Dijkstra algorithm [16] with cost $O(n^2)$, where $n$ is the total number of brokers. Let us revisit the example in Fig. 6, where the TRT of all brokers as well as SPRT of Broker B are presented. In this example, Broker B is connected to Broker H through F and G, and the optimal next hop to Broker H from Broker B is Broker F.

Message Processing: The multicasting layer provides a simple and efficient destination-based one-to-many message delivery service. At this layer, advertisements, subscriptions and events are routed in the same way. Each message is delivered to its destinations along the shortest paths. And, once a message is received by a destination broker, the message’s destIDList is reduced. This guarantees that each message is delivered to its destinations once and only once. As a result, general overlay topologies can be supported without redundant messages resulting from the broadcasting of advertisements.

Algorithm 2: Message Processing in the Multicasting Engine

1: for m ← input.dequeue() do
2:   if m is from a local client then
3:     match.enqueue(m)
4:   else
5:     if localBrokerID ∈ m.destIDList then
6:       match.enqueue(m)
7:     m.destIDList.remove(localBrokerID)
8:     Multicast(m)
9:   end for
10:   m.destIDList.remove(localBrokerID)
11:   Multicast(m)
12: end for

Fig. 8:

Alg. 8 processes messages from the network (stored in the input message queue) and messages generated by the matching engine (stored in the output message queue). In this algorithm, a message from a local client is simply inserted into the match queue for later processing by the matching engine. For a message from other brokers, if the broker is not one of its destinations, that message is simply forwarded. Otherwise, one copy of that message is inserted into the match queue. A message generated by the matching engine is first delivered to its destination clients and then forwarded to its destination brokers.

SendTo is a service provided by the underlying communication interface to deliver a message to a local client or a neighboring broker. Multicast is a function provided by the multicasting engine to forward a message to other brokers. Upon receiving a message with the destination list destIDList, the message is duplicated into several messages, and each one is attached with a new destination list destIDList:

\[
\text{nextID} \leftarrow \text{ShortestPath(destIDList)}
\]

where nextID is the optimal next hop from the local broker to all destination brokers in destIDList. To implement the function ShortestPath, every destination in destIDList is used to retrieve the optimal next hop from SPRT. Then destinations with the same optimal next hop are merged into a single destination list. Finally, each duplicated message is sent to its optimal next hop using the SendTo service.

C. Dynamic Overlay Reconfiguration

Since content-based matching and destination-based multicasting are decoupled, changes to the overlay will not impact the routing tables at the matching layer in the D-DBR algorithm. For example, in Fig. 6, if the connection between brokers B and F is lost, only brokers’ TRTs and SPRTs at the multicasting layer need to be updated. As a result, D-DBR can easily support dynamic overlay reconfiguration enabling fault-tolerance and performance optimization.

Fault Tolerance: In D-DBR, brokers use a heart-beat mechanism to detect the status of their neighbors. A broker multicasts an overlay update message to other brokers when a neighbor’s status changes. The brokers receiving the message update their TRTs and SPRTs accordingly. In this way, D-DBR can automatically recover from broker failures and disconnections. In Fig. 6, when the connection between brokers B and F is lost, the TRT and SPRT of Broker B would be automatically updated. Then the events from Broker A to G can be transferred through Broker E. In networks with cycles, unless there are no connecting routing paths between the source and the destinations, messages can always be successfully delivered using the latest optimal paths.

Performance Optimization: In pub/sub, the publication and subscription workload may vary over time. Capability to dynamically adapt the overlay to enable performance optimization is very important. In this section, we propose a simple distributed topology self-organizing algorithm for D-DBR. Its basic idea is to set up connections between brokers which incur heavy communication workload. In this algorithm, each broker periodically undergoes the following three stages:

1) Collection. During a performance optimization cycle $T^3$, each broker collects its communication rates. The communication rate between brokers $B_i$ and $B_j$, $rate_T(B_i, B_j)$, is defined as the number of messages transmitted from $B_i$ to $B_j$ or vice verse divided by $T$. In D-DBR, since messages are routed based on their destinations, the communication rate between any two brokers can be easily obtained.

2) Evaluation. Based on the communication rates, broker $B_i$ computes the gain of a candidate link to a non-neighbor broker $B_r$ and the lose of an existing link to a neighbor broker $B_n$. Their definitions are as follows:

\[
\text{gain}(i, r) = rate_T(B_i, B_r) \times (\text{dist}(B_i, B_r) - 1)
\]

\[
\text{lose}(i, n) = rate_T(B_i, B_n) \times (\text{dist}'(B_i, B_n) - 1)
\]

In above two formula, $\text{dist}(B_i, B_r)$ represents the current distance between brokers $B_i$ and $B_r$ in terms of the number of hops on the optimal path between $B_i$ and $B_r$, whereas $\text{dist}'(B_i, B_n)$ represents the distance between $B_i$ and $B_n$ after the connection between them is lost.

Candidate links to non-neighbor brokers are ranked based on their gains, and existing links are ranked based on their loses. Based on the rank and a configuration parameter MaxDegree\(^4\), each broker communicates with its neighbors to decide which new links can be added and which old ones should be discarded: First, a broker, say $B_r$, computes its minimal lose $\text{MinLose}(i)$ to free up a quota for a new link. $\text{MinLose}(i)$ equals zero if its current link number is less than MaxDegree, otherwise equals to the minimal lose of its existing links. Second, $B_i$ selects the candidate link $l(i, j)$ with the largest gain. Third, if $\text{gain}(i, j) > \text{MinLose}(i)$, $B_i$

---

\(^3\)T can be configured based on application requirements.

\(^4\)MaxDegree indicates the total number of links each broker can maintain.
communicates with \( B_1 \) to get its current \( \text{MinLose}(j) \). Finally, if \( \text{gain}(i, j) > \text{MinLose}(i) + \text{MinLose}(j) \), the link \( l(i, j) \) will be added and other related links will be discarded.

3. \textbf{Execution.} The related brokers update their connections, and a topology update message is issued by Broker \( B_1 \) to notify all other brokers to update their TRTs and SPRTs accordingly.

\textbf{D. Algorithm Analysis}

\textbf{Subscription Duplication:} D-DBR does not redundantly duplicate subscriptions across all brokers on the routing paths. Instead a subscription is only stored at its local broker and the brokers with matched advertisements. For example, in Fig. 6, the subscription \( s_3 \) is stored at brokers G and A only, not at the intermediate brokers, such as B and F.

\textbf{Matching Overhead:} The matching overhead in D-DBR is reduced for three main reasons: First, the size of PRT is smaller and the matching cost at each source broker is thus smaller. Second, an event is not matched at any intermediate broker. Third, an event only needs to be matched against the local subscriptions at each destination broker. Since event matching is the most expensive operation in pub/sub, reducing matching cost significantly alleviates the overhead of brokers and improves the pub/sub system’s performance.

\textbf{Flexibility and Robustness:} D-DBR efficiently supports a general overlay with cycles. In addition, since the content-based matching and destination-based multicasting are totally decoupled, D-DBR achieves good flexibility and robustness. It has good fault tolerance capacity and supports dynamic overlay self-reconfiguration.

\textbf{Routing Accuracy:} Since messages are routed on the shortest paths, D-DBR has a routing accuracy as good as that of the FBR algorithm. In some scenarios, where the loads between brokers are not balanced, D-DBR’s flexibility makes it possible to automatically reconfigure the overlay. As a result, in these scenarios, the lengths between the source and destination brokers can become shorter, and D-DBR can achieve better routing accuracy.

\textbf{Destination List Overhead:} An important factor that may limit the applicability of D-DBR is the destination list overhead. If the average destination list is long, more bandwidth is used. However, since a message’s destination list can be reduced at every routing step (by a factor of the fan-out of brokers in the message’s delivery tree), the destination list overhead is in fact very small. We will demonstrate this point through real world experiments and detailed simulations in Section V. On the other hand, events in content-based pub/sub systems often carry rich content. For example, in RSS applications, most feeds/events range from 1KB to 10KB, with a median of 5.8 KB [21]. In these scenarios, the destination list overhead occupies only a small fraction of bandwidth.

\textbf{Topology Maintenance Overhead:} In D-DBR, a broker needs to know all other brokers in the overlay instead of its own neighbor brokers. This introduces some extra overhead for topology maintenance. However, in small and medium-sized networks, the extra overhead is acceptable: heart-beat messages are only exchanged between neighboring brokers, a topology update message only needs to be delivered to a limited number of brokers, and the size of TRT and SPRT is not very large. For example, in an overlay with 100 brokers, the TRT and SPRT of each broker only occupies 11 kB of memory.

D-DBR is effective for small and medium-sized overlays. For large-scale overlays, however, the topology maintenance cost can be expensive. In a topology with 1000 brokers, the size of each broker’s TRT is 1000,000. Moreover, when a broker is near the source broker of events, the destination lists can be very long. In a topology with 1000 brokers, an event can at most carry 999 destination addresses. If lots of events are issued at the same broker, that broker’s bandwidth can be impacted. To address these limitations, we propose another solution that builds on the benefits of D-DBR while solving the problems related to the overhead of topology maintenance and large destination lists.

\section{MERC Design}

For better scalability and system management, we further propose a novel routing scheme called MERC—Match at Edge and Route intra-Cluster. MERC combines D-DBR and FBR hierarchically. It has the advantages of D-DBR, i.e., low subscription duplication, low matching cost, etc. It also overcomes the scalability limitation of D-DBR: In MERC, each broker needs to know a limited number of brokers only. As a result, the topology maintenance overhead and the destination list overhead are thus both reduced.

In MERC, the broker overlay is divided into interconnected clusters of brokers. Some brokers, called edge brokers, are located at the edge of clusters and belong to more than one cluster, whereas the other brokers, called internal brokers, belong to only one cluster. Each broker only knows the addresses of brokers in clusters it belongs to. FBR and D-DBR are adopted for inter- and intra-cluster event routing, respectively.

![Event routing in MERC](image)

\section*{Fig. 9: Event routing in MERC}

In MERC, when an event is issued, it is first matched against subscriptions at the local broker to identify interested brokers in the local cluster. Then the event is delivered to these brokers along the shortest paths, according to D-DBR. Once an event is received by an edge broker that also belongs to another cluster, the event is matched against subscriptions from that cluster at the edge broker to identify interested brokers in that cluster. It is then delivered to these brokers from the edge broker, again, according to D-DBR. This process is repeated.
until the event is delivered to all interested brokers in all clusters.

Fig. 9 shows an example of event routing in MERC. In this example, the broker overlay is divided into two clusters: Cluster A and Cluster B. Both are connected by an Edge Broker AB. There are two advertisements and three subscriptions: \( a_1, a_2, s_1, s_2, \) and \( s_3 \). In the example, we assume each advertisement matches all subscriptions. Now, when an event is issued at Broker \( A_1 \), it is first delivered to the interested brokers \( A_3 \) and \( AB \) in Cluster A. At the Edge Broker AB, it is matched against subscriptions from Cluster B and further transferred to the interested brokers \( B_2 \) and \( B_6 \) in Cluster B.

A. Routing Tables

In MERC, routing tables of internal brokers at both the matching layer and the multicasting layer are the same as those in D-DBR: Each broker maintains the same six routing tables: L-SRT, R-SRT, L-PRT, R-PRT, TRT and SPRT. However, the edge brokers have different routing tables. Besides a L-SRT and a L-PRT, an edge broker maintains a group of the other four routing tables for each cluster it belongs to. Note that in MERC, the sourceID of advertisements and subscriptions transferred by an edge broker from one cluster to another cluster is replaced by that edge broker’s ID.

For example, Fig. 9 shows that the Edge Broker AB maintains two R-SRTs and two R-PRTs, each of which is identified by the cluster’s name. Also, the sourceID of advertisements and subscriptions that are transferred by Broker AB from one cluster to another cluster is replaced by Broker AB’s ID.

B. Message Processing

In MERC, message processing for internal brokers is the same as that in D-DBR. For edge brokers, they override the function \textit{Forward} in Alg. 7\(^3\) to process messages at the matching layer with the new implementation shown in Alg. 10.

Fig. 10: Algorithm 3: Implementation of function \textit{Forward}(m)

\begin{verbatim}
1: if localBroker is an edge broker then
2: m.sourceID ← localBrokerID
3: if m is an advertisement then
4: m.destIDLList ← IDs of all brokers in other clusters
5: else if m is a subscription then
6: advs ← Match(m, R-SRTs of other clusters)
7: end
8: m.destIDLList ← advs.sourceID
9: else if m is an event then
10: subs ← Match(m, R-PRTs of other clusters)
11: m.destIDLList ← subs.sourceID
12: end
13: output.enqueue(m)
\end{verbatim}

Whenever a message from a specific cluster is delivered to an edge broker, the sourceID of that message is first replaced by that edge broker’s ID. Then that message is processed based on its type: An advertisement is forwarded to all brokers in other clusters, a subscription is forwarded to brokers in other clusters with matching advertisements, and an event is forwarded to brokers in other clusters with matching subscriptions.

For edge brokers, message processing at the multicasting layer is similar to that in D-DBR except that several groups of routing tables (TRT and SPRT) may be used to route a message to its destinations.

C. Algorithm Analysis

Scalability: Compared with D-DBR, MERC achieves better scalability: First, a broker only needs to know brokers in the cluster or clusters it belongs to. Changes in one cluster will not affect brokers in other clusters. Topology maintenance overhead for each broker remains small, even when the overlay’s size increases. Second, a message will not be annotated with addresses of brokers in other clusters. The size of each message’s destination list is thus limited by the number of brokers in a cluster.

Performance: In MERC, subscriptions are stored at the local broker, destination brokers, and some edge brokers. Events are matched at these brokers. The performance of MERC is lower than D-DBR since an event needs to be matched at some edge brokers. However, because an event needs not to be matched at any internal intermediate broker, its performance is still higher than FBR.

Other Considerations: Today’s Internet can be viewed as a collection of interconnected routing domains [9], which are groups of nodes that are under a common administration and share routing information. MERC follows this design: One cluster can be viewed as an administrative domain and different clusters can be connected in a hierarchical manner. In MERC, if an edge broker acts as a transit node, and an internal broker acts as a stub node, the pub/sub system follows the Transit-Stub model [35], which correlates with Internet structure. So an appealing characteristic of MERC is that it provides a good reference to construct large pub/sub systems following the structure of Internet.

V. EVALUATION

This section evaluates our D-DBR and MERC algorithms using experiments run on a computing facility and experiments based on simulations. We use the FBR algorithm\(^6\) as a baseline. In our experiments, we evaluate system throughput, event delivery latency, and overhead, measured as CPU utilization, destination address list size, subscription duplication, etc..

A. Experiments on Computing Facility

We implemented the D-DBR and MERC algorithms in PADRES [28], a representative, open-source, content-based pub/sub system implementing the FBR algorithm. We use the SciNet computing facility [26] as a testbed, in which each node has 8 cores at 2.66 GHz CPU and 8 GB of memory. In our experiments, each broker is deployed on one core. The TCP/IP communication links between the brokers represent the topology of the pub/sub system we deploy. In our experiments, we consider two representative topologies, i.e., an acyclic linear topology and a general topology generated by the GT-ITM network graph generator [35]. We apply stock quote trace datasets from Yahoo! Finance and synthesize subscriptions according to a Zipf distribution, which has been shown to be representative for the popularity of subscriptions on the Internet [21].

Acyclic Linear Topology: We first investigate the basic characteristics of D-DBR and MERC using an acyclic linear topology, which is simple and intuitive, and allows us to

\(^6\)The original FBR algorithm was designed for acyclic overlay networks only [22]. For general overlay networks, we employ the extension to FBR proposed in [19].
control the number of intermediate brokers through which each event travels. In the experiments, varying numbers of brokers are interconnected in a straight line. One publisher and 100 subscribers are connected to the broker at the head and the tail of the line, respectively. For MERC, the linear topology is divided into two clusters with the same number of brokers: The first half brokers are in the first cluster and the second half brokers are in the second cluster. The middle broker in the line serves as the edge broker for both clusters.

We measure the average event delivery latency and the CPU utilization with varying numbers of brokers and subscriptions. Event delivery latency is obtained by measuring the interval between the time when an event is issued by the publisher and the time when the event is received by the subscribers.

Fig. 11 to Fig. 13 show the results of three groups of experiments. In each group of experiments, the number of brokers increases from 1 to 10. We select 100, 800, and 2000 subscriptions to represent the scenarios with a small, middle and large number of subscriptions, respectively. The publishing rate is set to 3000 messages/minute.

Fig. 11 shows that given 100 subscriptions, event delivery latencies of FBR, D-DBR and MERC all increase as the number of brokers increases. The latency of D-DBR and MERC is smaller than that of FBR. However, as shown in Fig. 12, when there are 800 subscriptions, the event delivery latency of FBR increases rapidly with increasing number of brokers, whereas the latencies of D-DBR and MERC only increase slightly. When the number of subscriptions increases to 2000, the advantage of D-DBR and MERC is more significant. We can observe that the event delivery latency of MERC is roughly 1.5 times that of D-DBR. In these experiments, each event is matched three times in MERC and matched twice in D-DBR. The results indicate that the number of matches is an important factor contributing to the event delivery latency.

Fig. 14 presents ten brokers’ CPU utilizations when there are 2000 subscriptions in the system. For FBR, every broker’s CPU utilization is about 63%; for D-DBR, only the first and the last broker’s CPU utilization is about 63%, the remaining brokers’ CPU utilizations are about 2%; for MERC, only the first, the last and the middle broker’s CPU utilization is about 63%. These experimental results show that, in an acyclic linear topology, D-DBR and MERC achieve better performance than FBR when the number of subscriptions is large.

General Topology: To investigate the performance of our algorithms in general topologies, we constructed an overlay of 100 brokers for FBR and D-DBR, where each node has an average node degree equal to 6. For MERC, we constructed an overlay of 100 brokers following the transit-stub model [35]. There are 10 transit nodes that act as edge brokers and 90 stub nodes that act as internal brokers. In total, there are 10 clusters. Here, the average node degree is also equal to 6. In this experiment, unlike internal brokers, each edge broker is deployed on 8 cores. For FBR, D-DBR, and MERC, every broker has an attached publisher and subscriber, which issues 1 advertisement and 30 subscriptions respectively.

Fig. 15 compares the performance of FBR, D-DBR and MERC. The results show that D-DBR has the best performance and MERC lies between D-DBR and FBR. When the event publishing rate (messages/minute) increases from 2000 to 2500, the event delivery latency for FBR increases from 1087 ms to 2843 ms. However, the event delivery latency for D-DBR is only 491 ms when the event publishing rate is at 14,000. Therefore, compared with FBR, D-DBR improves the throughput by up to 700% and reduces the communication latency by up to 55% at the same time. We also found that the latency for MERC is stable and slightly higher than D-DBR when the event publishing rate is less than 11000. This suggests that powerful edge brokers are provided, MERC also performs much better than FBR.

In the experiments, we recorded the messages received and sent by each broker and computed the average destination list overhead for each broker. Fig. 16 shows that the average destination list size at each broker ranges from 1 to 5 for D-DBR and 1 to 1.6 for MERC. On average, MERC has smaller destination list size than D-DBR (1.18 vs. 2.12).

In FBR, advertisements need to be broadcast. In this experiment, even though only 100 advertisement messages are issued, more than 38000 duplicated advertisement messages are detected. In D-DBR and MERC, advertisements are routed in the same way as events. As a result, no duplicated advertisement messages are generated.

Fig. 17 shows the number of duplicated subscription for each issued subscription when the ratio of brokers with matching advertisements varies. D-DBR has ideal subscription duplication feature, which means a subscription is only duplicated at brokers with matched advertisements. MERC also presents very good subscription duplication feature. However, for FBR, subscriptions are heavily duplicated. For example, when 80% of brokers has matched advertisements, each subscription is duplication for 221 times, on average7.

Fig. 18 shows the number of subscriptions maintained by each broker when 60% of the brokers have matched advertisements. Overall, the 100 subscribers issue 3000 different subscriptions, each broker stores 5005, 1800, and 1945 subscriptions for FBR, D-DBR, and MERC respectively.

B. Experiments Based on Simulations

Destination List Overhead: This section investigates the destination list overhead of the algorithms as the number of brokers, the average broker degree, and the ratio of interested brokers for each event changes. In our studies, the source broker and destination brokers for each event are randomly selected. For each configuration of the factors, we simulate D-DBR routing for 10000 times using the topologies generated by GT-ITM network graph generator.

We generated different topologies with a broker degree equal to 6. Fig. 19 shows that the average destination list size grows when the number of brokers increases. In addition, the more brokers are interested in an event, the larger the average destination list becomes. Nevertheless, the average destination list size is quite small, even in a large overlay. For example, in an overlay with 700 brokers, even though an event is issued to all brokers, each message carries the IDs of only 3.8 brokers, on average. The destination list size can be further reduced by increasing the brokers’ degree (e.g., in the aforementioned.

7To support FBR in general overlay, a broker may need to store many copies of the same subscription [19].
example, each message carries the IDs of only 3.2 brokers, on average, if the average broker degree is 9).

Fig. 20 shows the distribution of destination list size and its cumulative distribution in a topology with 700 brokers. The average broker degree is set to 6 and an event is delivered to 10% of the brokers. The results show that more than 95% of the destination list sizes are smaller than 5 and more than 99% of them are smaller than 12.

When it is close to the source broker of an event, the generated event messages may carry long destination lists. We study this overhead by computing the average destination list size at source brokers for all events. As shown in Fig. 21, the destination list size increases linearly as the network scales up. Moreover, the larger the ratio of interested brokers, the longer the destination list becomes. Therefore, in large-scale networks, if a broker issues many events to a large number of other brokers, its destination list overhead can be significant.

MERC reduces the destination list size because messages are annotated with the addresses of brokers in the local cluster only. For example, in a topology with 1000 brokers, divided across 10 clusters, the destination list size for MERC is equal to that of D-DBR with 100 brokers only.

**Routing Accuracy:** In D-DBR, the overlay topology can be dynamically adjusted for performance optimization purposes. This can improve its routing accuracy, as we will show in this experiment.

In an overlay with 70 brokers, the workload between every two brokers is randomly generated within the range of 1 to 100 messages/minute. We computed the number of hops each message travels for this overlay. Then, we dynamically reconfigure the overlay, using the solution proposed in Section III-C, and computed the number of hops each message travels for the new overlay.

Fig. 22 shows that after the overlay reconfiguration, the number of hops each message travels is reduced. Overall, the average number of hops for all messages is reduced from 3.97 to 3.51, which means the routing accuracy is improved from 0.252 to 0.285.

**System Robustness:** To study the system robustness of FBR and D-DBR, we compare the successful event delivery rate when the system is subjected to broker failures and connection failures.

![Fig. 23: Robustness under broker failures](image1)

![Fig. 24: Robustness under connection failures](image2)

![Fig. 25: Throughput and latency](image3)

![Fig. 26: Destination list size distribution](image4)

![Fig. 27: Subscription duplication distribution](image5)

![Fig. 28: Substitution number distribution](image6)

![Fig. 29: Average destination list size](image7)

![Fig. 30: Average destination list size distribution](image8)

![Fig. 31: Average destination list size at source brokers](image9)

![Fig. 32: Routing accuracy](image10)
ratio when brokers crash and their connections are lost. In our experiments, the overlay has 100 brokers and the average broker degree is 6. Each broker is attached with one publisher and one subscriber. A subscriber issues one subscription and a publisher publishes one event. We assume each event should be delivered to every subscriber.

Fig. 23 shows that D-DBR is more robust than FBR under broker failures. For example, when 50% brokers fail, 21.8% events are successfully delivered using D-DBR, whereas only 8.7% events are successfully delivered using FBR. Fig. 24 shows that D-DBR is also more robust than FBR for connection failures. For example, when 50% connections are lost, 86.2% events are successfully delivered using D-DBR, whereas only 17.5% events are successfully delivered using FBR.

MERC applies the D-DBR algorithm for intra-cluster event routing and thus inherits D-DBR’s flexibility and robustness. However, the failure of an edge broker may cause a whole cluster to become disconnected from the system in the worst case. Therefore, MERC may suffer from single points of failure caused by edge brokers. We plan to investigate this issue in future work.

**Topology Maintenance Overhead:** The topology maintenance overhead for D-DBR and MERC consists of two parts: The memory to store TRT and SPRT tables and the network overhead for overlay reconfiguration. In D-DBR, the scale of TRT and SPRT are equal to $N^2$ and $N$ respectively, where $N$ is the number of brokers in the system. $N$ messages are needed for overlay reconfiguration. For MERC, if the overlay is evenly divided into $M$ clusters, the scale of TRT and SPRT are equal to $\frac{N^2}{M^2}$ and $\frac{N}{M}$, and only $\frac{N}{M}$ messages are needed for overlay reconfiguration.

**VI. CONCLUSION AND FUTURE WORK**

In this paper, we presented the design and the evaluation of D-DBR, an efficient algorithm for content-based pub/sub, and MERC, a routing scheme to extend D-DBR to a hierarchical structure to efficiently operate at greater scale. Both algorithms exhibit some important advantages over existing approaches that materialize in improvements of performance, flexibility, robustness, and scalability.

D-DBR exhibits low processing overhead by reducing the number of event matching computations and exhibits higher flexibility by decoupling event matching from event routing at the cost of requiring global topology knowledge and the requirement to carry destination lists in messages. D-DBR is well-suited for small and middle scale networks. For larger scale networks, MERC is more suitable, because each broker needs knowledge of a small part of the overall network topology only, and, consequently, the destination list overhead is reduced. Moreover, brokers in MERC are organized as a structured and hierarchical network to provide better system management and maintenance. The limitation is that the edge brokers in MERC may suffer from single points of failure and need more powerful machines.

Our experimental results show that D-DBR and MERC perform well across a range of scenarios with significant performance improvements, especially, when there are a large number of subscribers.

**REFERENCES**


