Adapting Publish-Subscribe Routing to Traffic Demands

Matteo Migliavacca
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
migliava@elet.polimi.it

Gianpaolo Cugola
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
cugola@elet.polimi.it

ABSTRACT

Most of currently available content-based publish-subscribe systems that were designed to operate in large scale, wired scenarios, build their routing infrastructure as a set of *brokers* connected in an acyclic network. The topology of such network is critical for the performance of the system. Depending on the traffic profile, the same topology may provide good performance or be very inefficient. Starting from this consideration, in this paper we first analyze this issue in detail, then we describe a distributed algorithm to address it, by adapting the topology of a content-based publish-subscribe routing network to the application demand.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems

General Terms

Algorithms, Performance

Keywords

Optimization, publish-subscribe Systems, Content-Based Routing, Adaptive Routing, Autonomic Computing

1. INTRODUCTION

Large scale publish-subscribe applications are usually built around an *overlay dispatching network* composed of a set of interconnected *brokers* that cooperate to provide the publish-subscribe service [17]. Each component of the application is connected to one of such brokers that acts as its entry point to the dispatching network. Through this broker, the component may publish messages and receive the messages it subscribed to. This interaction among the brokers and the components that build the publish-subscribe application explains why the latter are usually called *"clients"*.

This work was partially supported by the italian National Research Council (CNR) under the IS-MANET project, and by the European Community under the IST-034963 WASP project.

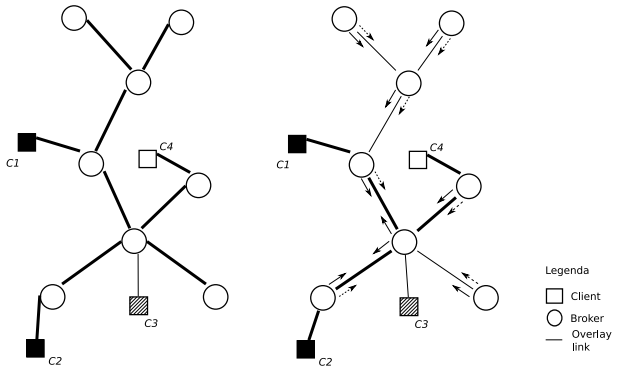

Figure 1: Routing on an acyclic overlay. Clients $C1$ and $C2$ are subscribed to the same "filled" filter, while client $C3$ is subscribed to a "dashed" filter. Where present, arrows represent the content of brokers' routing tables, while thick lines represent the links followed by a message sent by client $C4$, which matches the filled filter.

Most of the existing publish-subscribe systems designed to operate on large scale, wired networks (usually IP networks) connect all brokers in a single acyclic overlay (i.e., an unrooted tree) [17, 16]. This is true, in particular, for *content-based* systems, where subscriptions are expressed as predicates (usually called *filters*) on the content of messages. Such systems, in fact, lack any explicit notion of "group" or "subject", as in traditional multicast or in subject-based systems[1]. The dispatching network must compute the set of recipients of each message by looking at its content and at the set of subscriptions known at publishing time. This makes it impossible to use separate dispatching trees for each group, as it happens in traditional multicast and subject-based systems. Conversely, by joining all the brokers in a single tree, messages can be easily routed, either by sending each published message to all the brokers and letting them to compute the set of clients interested in the message (see Figure 1, left), or by propagating information about clients' subscriptions along the same dispatching tree and subsequently using such information to route messages only toward interested brokers, i.e., those that have at least one subscribed client attached to them (see Figure 1, right).

As the first type of routing does not scale well when the

[1]Also known as "topic-based" systems.

size of the system grows, most of the proposals in the area of content-based publish-subscribe focus on the second form of routing, dispatching messages only toward interested brokers along a single acyclic overlay. At the same time, even this strategy has a potential weakness: the cost of routing depends on the topology of the dispatching network, which is usually defined at deployment time and never changes. Indeed, if the publisher of a message m is connected to the opposite side of the routing network w.r.t. the subscribers of m, the message must traverse a large number of brokers to reach its recipients. Even worse, if the subscribers are very sparse w.r.t. each other, m must flood the entire dispatching network.

In this paper we propose a distributed algorithm to overcome the limitation above. It uses information about the messages effectively routed by the publish-subscribe system to periodically reconfigure the dispatching network at runtime to reduce the overall routing cost.

The paper is organized as follows. Section 2 models publish-subscribe routing and formally states the problem we address, studying its complexity in general terms. Section 3 describes the reconfiguration algorithm we propose, while Section 4 place our work in the context of related research efforts. Finally Section 5 provides some concluding remarks and illustrates future work.

2. PROBLEM STATEMENT

As mentioned in the previous section, the problem we address in this paper can be defined informally as: *to adapt the topology of an acyclic content-based publish-subscribe routing network to the application demand, minimizing the routing cost.* We call this problem OCBR, an acronym which stands for Optimal Content-Based Routing.

To formalize the OCBR problem, we model a publish-subscribe system as a set of brokers B and clients C, building a routing network $Net = (N, L)$, where $N = B \cup C$ is the set of network nodes, while L is the set of available network links. We assume symmetric links, so the graph *Net* is undirected, and we use the notation $\langle n_1, n_2 \rangle$ to designate the link that joins node n_1 with node n_2. Moreover, since the clients of a publish-subscribe system always connect to brokers and never connects directly among them, we assume that the set L does not include links that directly joins two clients. Formally: $\forall c_1, c_2 \in C\ (\langle c_1, c_2 \rangle \notin L)$. Finally, observe that, if the publish-subscribe system we consider builds its routing infrastructure as a true overlay network (e.g., on top of an IP infrastructure) the set L includes all the links that joins every broker with every other and every client with every broker, otherwise only some of those links are available, e.g., when the content-based routing infrastructure is built at the networking layer (in the OSI jargon), with brokers joined by data-link layer connections.

To model the application demand, we assume that given a set of n clients: $c_1, \ldots, c_n$, the cost of routing a message published by one of them to all the others does not depend on the choice of the publisher. In other words, the cost of routing a message from c_1 to $c_2, \ldots, c_n$ equals the cost of routing a message from c_2 to $c_1, c_3, \ldots, c_n$, and so on. This assumption is easily verified if links are symmetric and messages are routed following the same tree spanning the publisher and the subscribers, independently from the identity of the publisher itself.

Given this assumption, we model the application demand as a couple (T, d), where $T \subseteq \mathbb{P}(C)$ is a set of *traffic classes* and $d : T \to \mathcal{N}$ is a function that maps each traffic class $t = \{c_1, \ldots, c_{n_t}\}$ to its *traffic demand* $d(t)$, expressed as the total number of messages published by any of $c_1, \ldots, c_{n_t}$ and delivered to all the others, and only to them.

As for the cost of routing a message, it can be decomposed into a cost associated to the traversal of links, plus a cost of matching associated with brokers. We model these costs through two functions: $C_l : L \to \mathcal{N}$ models the average cost of sending a message through a link, while $C_b : B \to \mathcal{N}$ models the average cost borne by a broker to process a message.

To calculate the total cost of routing a single message for a traffic class t, we need some information about the way messages are routed. In particular, we assume a topology that joins brokers in a single spanning tree and each client to one and only one broker. We call *valid* a spanning tree on *Net* that satisfies such constraints. On top of a valid topology, we assume a routing strategy that behaves as in Figure 1, right, by routing messages along the minimal subtree that connects the publisher with all the subscribers. Under these assumptions, which, as mentioned in Section 1, are satisfied by most of the existing content-based publish-subscribe systems, the problem we stated informally at the beginning of this section can be formalized as follows:

OCBR problem
given a routing network *Net*, a set of traffic classes T, with associated demands d, and two cost functions C_l and C_b
find a valid spanning tree ST on *Net*
minimizing the routing overhead:

$$\sum_{t \in T} \left(\alpha \sum_{l \in E(ST_t)} C_l(l) \cdot d(t) + \beta \sum_{b \in (V(ST_t) \cap B)} C_b(b) \cdot d(t) \right) \quad (1)$$

where ST_t is the minimal subtree of ST including the clients in t, while $E(ST_t)$ and $V(ST_t)$ are the set of links and nodes in ST_t, respectively.

The two parameters α and β must be chosen to find the required compromise between minimizing the overall cost associated with links and that associated with brokers.

It is easy to see that the optimization problem above is not solvable in polynomial time. To prove this fact consider the Steiner tree problem [13] below:

Steiner tree problem
given a graph $Net_{\text{stein}} = (B_{\text{stein}}, L_{\text{stein}})$, a cost function $C_{\text{stein}} : L_{\text{stein}} \to \mathcal{N}$, and a subset of *terminal* nodes $t_{\text{stein}} \subseteq B_{\text{stein}}$
find a minimal cost tree ST_{stein} spanning all the terminals t_{stein}.
If we pick:

- a network $Net = (B_{\text{stein}} \cup C_t, L_{\text{stein}} \cup L_t)$, where C_t is a set of $|t_{\text{stein}}|$ clients and L_t is the set of $|C_t|$ links connecting each client in C_t with a different broker in t_{stein};
- a single traffic class $t_0 = C_t$, with demand $d(t_0) = 1$;
- a cost function C_l such that: $\forall l \in L_{\text{stein}} | C_l(l) = C_{\text{stein}}(l)$, while $\forall l \in L_t | C_l(l) = 0$; and
- $\alpha = 1$ and $\beta = 0$;

we easily see that the resulting OCBR problem solves the Steiner tree problem above. More specifically, under the

conditions above, the cost function (1) we consider coincides with the cost of the tree spanning all the terminals t_{stein}. This shows that it is possible to reduce any Steiner tree problem to a specific OCBR problem, which, consequently, must be at least as complex to solve as the Steiner tree one.

At the same time, the approach taken in the proof above, showing how the OCBR problem with a single traffic class is equivalent to the Steiner tree problem, suggests that the real OCBR problem, involving hundreds of traffic classes to be optimally organized on a single tree, is fundamentally different from the Steiner tree problem. Accordingly, approximate solutions available for the latter are not easily adaptable to the former.

3. THE ALGORITHM

Before going into the details of an algorithm capable of approximating the solution of the OCBR problem, we observe that the formalization given above does not specify how the application demand is obtained. While building our formal model it was reasonable to consider it a known input of the problem, now it is time to define how it can be obtained in practice.

Clearly, the demand of a content-based publish-subscribe application cannot be known before running it. On the other hand, our final goal is to implement a publish-subscribe system capable of adapting its routing infrastructure to application demands *at run-time*. Consequently, we need a way to forecast the application demand. We do so by collecting traffic information for a given time period Δ_t and by rearranging the dispatching network at the end of such time period to minimize the routing overhead *as if* the same traffic would repeat in the future. This "measure → rearrange" process is repeated every Δ_t.

This approach is reasonable as soon as the overall pattern of application traffic changes slowly during time, which seems to be a reasonable assumption for most publish-subscribe applications. We will not be able of finding the best solution for the actual demand but we will find a good approximation.

Unfortunately, this is neither the only problem we have, nor the most complex. Indeed, in the last section we shown that the OCBR problem cannot be solved in polynomial time even if a single traffic class is considered. In a real content-based publish-subscribe system the total number of traffic classes is much greater. Since $T \subseteq \mathbb{P}(C)$, such number can grow up to $2^{|C|}$. This is a typical consequence of content-based routing, where each message has not to be routed to a group of recipients chosen among a limited, predefined set of groups, as in subject-based systems [18]. Moreover, determining such traffic classes and the associated demands requires a global view of the network, something hard to obtain in practice.

In other words, even the problem of determining the inputs of the OCBR problem, i.e., the traffic classes with their associated demands, during the period of observation Δ_t, is intractable if we consider the entire network as a whole. We need a distributed solution.

3.1 Towards a distributed optimization algorithm

Fortunately, to solve the OCBR problem and overcome all the difficulties above, we may take benefit of its own distributed nature. Instead of looking for a solution to the problem as a whole, we restrict it to smaller portions of the publish-subscribe network and perform the optimization in parallel on separate subnetworks. In particular, since we operate on a tree shaped network, we consider the smallest possible sub-network: the tree composed of a broker and its immediate neighbors. Accordingly, our algorithm operates by letting each broker periodically rearrange its neighborhood to minimize (1); to avoid interference among concurrent reconfigurations involving the same brokers, we lock such neighborhood before rearranging it.

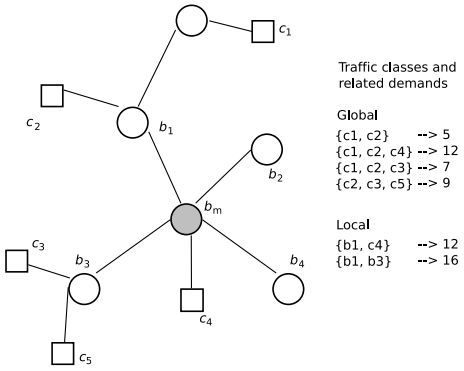

Figure 2: An example of the simplification of the OCBR problem coming from looking at local reconfigurations only.

More specifically, periodically each broker b_m, the *master* of the current *local reconfiguration* tries to minimize (1) by only considering the network composed of itself and its neighbors. This means that it collects the traffic classes it observes directly, i.e., those formed of its neighbors only, and locks and rearranges its neighborhood in a spanning tree that minimizes (1).

We can relate the traffic classes perceived from the master b_m viewpoint in terms of the network-wide traffic classes defined in the formalization given in the previous section, in fact

1. b_m does not perceive and is not interested into the traffic classes composed of clients that are all part of the same subtree rooted at one of its neighbors: in fact the related messages do not enter the portion of the system considered by b_m, thus cannot be affected by any b_m's neighborhood reconfiguration;

2. for each traffic class t among the remaining ones, and for each client $c \in t$ that is not directly attached to it, b_m view substitutes c in t with the broker at the root of the subtree that includes c[2]. This way b_m view "merge" several traffic classes together, further reducing their number.

As an example, consider the publish-subscribe system in Figure 2 and the related set of traffic classes with associated demands. When the gray broker b_m operates as a master of a local reconfiguration, it looks only at the subtree composed by itself and its neighbors: b_1, b_2, b_3, b_4, and

[2]In doing so we extend the definition of "traffic class" to include also the brokers as possible end-points of the communication.

c_4. Consequently, it may ignore the traffic class $\{c1, c2\}$, whose 5 messages do not flow through the subtree it considers. Also, it is not interested in (and usually it cannot determine) the specific source of the messages that come from the same neighbor. Accordingly, it perceives the traffic class $\{c1, c2, c4\}$ as $\{b1, c4\}$, and it merges the two classes $\{c1, c2, c3\}$ and $\{c2, c3, c5\}$ into a single class $\{b1, b3\}$. At the end, the total number of traffic classes to consider halves (from 4 to 2). In real scenarios, with thousands of traffic classes, the simplification can be much greater.

Before proceeding, we must evaluate how the local reconfiguration performed by a broker b_m by only looking at its neighborhoods impacts the global overhead. It is easy to see that, independently from the choice of the broker b_m, the improvement to the *local routing cost* it measures using the rules above equals the improvement in the *global routing cost* calculated by (1). This is a consequence of the additive nature of (1) and of the way the systems we consider route messages. This guarantees that an algorithm that repeatedly let each broker rearrange its neighborhood until none of them can find a better topology will eventually terminate: a fundamental feature of any optimization algorithm.

The last issue to consider is how each local reconfiguration is performed. As we noticed, operating locally greatly reduces the complexity of the problem to solve: the number of nodes involved reduces to the neighborhood of the master b_m and the number of traffic classes reduces accordingly. At the same time, the problem still remains np-hard. In particular, the number of possible configurations (i.e., spanning trees) that a master b_m of a local reconfiguration has to consider, depends on the number N of its neighbors and equals N^{N-2}. An exhaustive search is feasible only for very small neighborhoods. In real scenarios, when the number of neighbors of each broker grows, some heuristic is required: the next section describes indeed the one we adopted.

3.2 Reconfiguring the neighborhood efficiently

A general approach often used to approximate complex problems is to use a *local search technique*. It is composed of an initialization step, which consists of processing the input data for the problem and building a starting feasible solution, followed by three repeating steps:

1. defining a set of *moves* that bring from the current solution to a set of "similar" ones;
2. exploring them, evaluating their quality with respect to the objective function;
3. selecting and applying the best move.

These steps are repeated until the optimum is reached or a certain number of moves has been applied.

A specific local search technique is *Tabu* [12]. It applies the best move found at step 3 even if the resulting solution is worse than the current one. Afterwards, it inserts the selected move into a *tabu list*, which is consulted in further iterations to exclude moves that could bring to solutions already explored. This results in escaping local minima, thus more easily converging toward the best solution.

We apply the tabu strategy to explore the space of all the valid trees spanning the master of a local reconfiguration and its neighbors. The master b_m of a local reconfiguration builds a view of that part of the overall network *Net* that includes only itself and its neighbors, and determines the application demand that concerns this part of the network as explained in the previous section.

As shown in Figure 3, the `optimize` algorithm executed by b_m starts by initializing the `currentTopology`. At the beginning this is a star centered at the master. The moves we consider are those that add a link to the current topology and remove the link (among those that build the resulting loop) that, once removed, causes the greatest decrease in (1). Since the possible number of such moves can be too big to be tractable in a reasonable time, at each round we limit our analysis to N of them, with N being a parameter that balances the total time required to find a solution with the accuracy of such solution.

We determine the best move among the N identified at the step above by looking at the cost function (1), and use the chosen move to calculate a new configuration. At the end, we add the move just selected to the tabu list. In particular, we use two different lists. The `addedLinks` list is consulted when choosing the link to remove, to avoid removing a link just inserted. Similarly, the `removedLinks` list is used to avoid adding a link just removed.

We repeat this process a given number of times, ending with the best configuration found. If it is better than the starting "star" topology, we apply it to the real system, by reconfiguring the network composed of the master and its neighbors; otherwise we decide that no better configuration exists.

4. RELATED WORK

During the last years, several content-based publish-subscribe middleware have been developed, both by the industry and academia. After the first implementations, built around a centralized dispatcher, most of the research focused on improving scalability: a goal that has been achieved by distributing the dispatcher. Accordingly, today the large majority of content-based publish-subscribe systems designed to operate in large-scale, wired scenarios adopt a dispatching network composed of a set of brokers, which cooperate to route messages from publishers to subscribers. As acknowledged by several surveys of the area: [3, 11, 16, 17], most of these systems organize their dispatching network in an acyclic graph, which is exactly the assumption we started from. The main consequence of this fact is that the OCBR algorithm we propose can be easily applied to all these systems.

An exception to the acyclic topology we took as our starting point is represented by two systems: XNet [7, 8] and Hermes [20, 21], and by the routing protocols described in [5, 6]. While OCBR cannot be directly applied to these routing strategies, we think that it could be interesting to test whether they provide effective improvements with respect to a self optimizing, single-tree based strategy, as that realized by OCBR. We plan to answer this question in future research.

Other content-based publish-subscribe systems that adopt different routing strategies are those proposed for very dynamic, mobile scenarios like MANETs [9]. While these proposals are very interesting, they are out of the scope of OCBR, which was explicitly designed to optimize the overlay routing network of publish-subscribe systems for large scale, wired scenarios.

As for a direct comparison with other works, the problem of adapting publish-subscribe routing to traffic has been al-

```
// Net_local is the neighborhood of the master
// C_l and C_b are the costs associated to links and brokers, respectively
// T_local and d_local models the application demand seen by the master
// n is the number of iterations of the algorithms
optimize(Net_local, C_l, C_b, T_local, d_local, n) begin
    // Initial topology is a star centered at the current node
    currentTopology ← starTopology
    addedLinks ← ∅
    removedLinks ← ∅
    bestTopology ← currentTopology
    for n iterations do
        // Calculate a set of N possible moves
        moves ← ∅
        while |moves| < N do
            Let linkToAdd be a random link in Net_local but neither in currentTopology.links() nor in removedLinks
            Let linkToRemove ← bestLinkToRemove(currentTopology, T_local, d_local, C_l, C_b)
            if linkToRemove ∈ addedLinks then
                continue
            end
            Add ⟨linkToAdd, linkToRemove⟩ to moves
        end
        // Find the best move
        best ← best(moves, currentTopology, T_local, d_local, C_l, C_b)
        // Update tabu lists and currentTopology
        Add best.linkToAdd to addedLinks
        Add best.linkToRemove to removedLinks
        currentTopology.addLink(best.linkToAdd)
        currentTopology.removeLink(best.linkToRemove)
        // Update best solution found, if necessary
        if cost(currentTopology, T_local, d_local, C_l, C_b) < cost(bestTopology, T_local, d_local, C_l, C_b) then
            bestTopology ← currentTopology
        end
    end
    // If the best solution found is worth applying, apply it
    if cost(bestTopology, T_local, d_local, C_l, C_b) < cost(starTopology, T_local, d_local, C_l, C_b) then
        reconfigureNeighborhood( bestTopology)
    end
end
```

Figure 3: The main reconfiguration algorithm used by each broker to change its neighborhood.

ready faced in [2]. The approach adopted by the authors is quite different from that described here. First of all, the authors adapt the publish-subscribe routing tree on the basis of the similarity among the interests of each client, while OCBR considers the traffic effectively exchanged. As a result, our approach is more general (it does not depend on any specific subscription language), it is simpler (it does not require to compare subscriptions, which is usually a hard task), and it is more precise (it considers the messages effectively exchanged)[3]. Also, the proposed approach uses a very simple metric (the number of hops to route messages), which does not consider the effective cost of routing. Also, according to the way the algorithm works, there is no guarantee that it would converge. It may happen that two brokers are joined because they have similar interests but, to keep the graph acyclic, a link should be removed resulting in increasing the distance among other brokers that had also very similar interests. Conversely, every OCBR local reconfiguration is guaranteed to reduce the overall routing cost, i.e., the figure we want to minimize. Finally, while OCBR confines each step of the protocol (i.e., each "local reconfiguration") within the neighborhood of the master, the algorithm proposed in [2] involves, at each step, a much larger area of the network. This requires, if implemented in a real setting, to synchronize large part of the publish-subscribe network in order for the reconfiguration to happen without loosing messages.

Another work that addresses the reconfiguration problem we consider, is described in [14]. The author describes the very preliminary results of a research in the area, by providing an in depth analysis of the work described in [2], identifying all the weaknesses we mentioned above. Unfortunately, it does not provide any solution to them. To this end, OCBR represents the first comprehensive proposal to overcome most of them.

Two proposals exploiting mechanisms similar to our iterated local rearrangement to optimize dispatching tree overlays are [4]. and [19]. However both works are only applicable to hierarchical overlays (rooted trees). The work in [4] is in fact tailored to data streams for real-time applications, and supposing there is a single sender, it tries to optimize the minimum average-latencies from the clients (receivers). The work does not consider costs for processing messages on nodes and is limited to latencies for link costs, our approach being more general. The metrics optimized by [19] are instead very customizable. Applications can indeed define a *local cost* as combinations of both link and processing costs. Moreover the cost accounted for a node (called *node cost*) can be an aggregation (e.g. max, min, sum) of local costs towards the root, finally the *system cost* is, in turn, an aggregation of *node cost*s. This allows for sophisticated optimization metrics such as "minimum of the sum of link latency towards the root" (minimum path latency). However this metrics flexibility comes at a price: even in this case routing must be single source for these metric to make sense wrt the routing process, our OCBR algorithm instead considers the most general publish-subscribe scenario.

[3]A subsequent proposal by the same authors: [1], has recognized and removed this limitation.

5. CONCLUSIONS

Most of currently available content-based publish-subscribe systems organize their routing network as a set of brokers and clients connected in a single (unrooted) tree. This provides a good compromize between the complexity of content-based routing and its efficiency. At the same time, the choice of the topology of such tree is critical to minimize the overall routing effort.

In this paper, we addressed this issue by providing a model to measure the total cost to route a given set of messages. Using it, we show that determining the best topology to minimize the routing effort is a np-hard problem. Starting from this result, we described the distributed algorithm OCBR to adapt at run-time the topology of a content-based publish-subscribe routing network to the application demand, reducing the routing effort. We are currently completing the simulation of our algorithm to measure its benefits under realistic scenarios. The interested reader may refer to [15] for the preliminary results of such simulations, which were omitted from this paper, due to space limitations.

Our plan for the future is to complete the set of simulation and then to implement the OCBR algorithm into the open source REDS content-based publish-subscribe middleware [10]. This will enable a more direct assessment of its performance in real-world applications.

6. REFERENCES

[1] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. A self-organizing crash-resilient topology management system for content-based publish/subscribe. In *Proc. of the Int. Workshop on Distributed Event-Based Systems (DEBS04)*, Edinburgh, Scotland, UK, May 2004. IEEE Computer Society.

[2] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Subscription-driven self-organization in content-based publish/subscribe. In *Proc. of the 1st Int. Conf. on Autonomic Computing (ICAC04)*. IEEE Computer Society, 2004.

[3] R. Baldoni and A. Virgillito. Distributed event routing in publish/subscribe communication systems: a survey. Technical report, Dipartimento di Informatica e Sistemistica, Universit di Roma "La Sapienza", 2005.

[4] S. Banerjee, C. Kommareddy, K. Kar, S. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *INFOCOM*, 2003.

[5] A. Carzaniga, A. J. Rembert, and A. L. Wolf. Understanding content-based routing schemes. Technical Report 2006/05, Faculty of Informatics, University of Lugano, Sept. 2006.

[6] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, Mar. 2004.

[7] R. Chand and P. Felber. XNet: a reliable content based publish subscribe system. In *Proc. of the 23rd Symp. on Reliable Distributed Systems*, Oct 2004.

[8] R. Chand and P. A. Felber. A scalable protocol for content-based routing in overlay networks. In *Proceedings of the 2nd IEEE Int. Symp. on Network Computing and Applications (NCA03)*, page 123, Washington, DC, USA, 2003. IEEE Computer Society.

[9] G. Cugola, A. Murphy, and G. Picco. Content-based Publish-subscribe in a Mobile Environment. In P. Bellavista and A. Corradi, editors, *Mobile Middleware*. CRC Press, 2006. Invited contribution. To appear.

[10] G. Cugola and G. Picco. REDS: A Reconfigurable Dispatching System. In *Proc. of the 6th Int. Workshop on Software Engineering and Middleware (SEM06)*, pages 9—16, Portland, Oregon, USA, nov 2006. ACM Press.

[11] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 2(35), June 2003.

[12] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[13] F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem*. Elsevier, North-Holland, 1992. (Annals of Discrete Mathematics, vol. 53).

[14] M. A. Jaeger. Self-organizing publish/subscribe. In *Proc. of the 2nd Int. Doctoral Symp. on Middleware (DSM05)*, pages 1–5, New York, NY, USA, 2005. ACM Press.

[15] M. Migliavacca and G. Cugola. Adapting publish-subscribe to routing demands. Technical report, Politecnico di Milano, 2007. Available on-line at `http://www.dei.polimi.it/upload/cugola`.

[16] G. Mühl, L. Fiege, F. Gartner, and A. Buchmann. Evaluating advanced routing algorithms for content-based publish/subscribe systems. In *Proc. of the* 10^{th} *IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS02)*, 2002.

[17] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer, 2006.

[18] L. Opyrchal et al. Exploiting IP multicast in content-based publish-subscribe systems. In *Proc. of the* 2^{nd} *Int. Conf. on Middleware*, 2000.

[19] O. Papaemmanouil, Y. Ahmad, U. Çetintemel, J. Jannotti, and Y. Yildirim. Extensible optimization in overlay dissemination trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 611–622, 2006.

[20] P. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, Computer Laboratory, Queens' College, University of Cambridge, February 2004.

[21] P. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proc. of the 2nd Int. Workshop on Distributed Event-Based Systems*, June 2003.