Overlay Self-Organization for Traffic Reduction in Multi-Broker Publish-Subscribe Systems

Elisabetta di Nitto, Daniel J. Dubois, Raffaela Mirandola
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano, Italy
{dinitto, dubois, mirandola}@elet.polimi.it

ABSTRACT

Publish-subscribe systems are characterized by two types of components: publishers and subscribers. Publishers publish messages without knowing the recipients, subscribers subscribe to some kinds of messages and receive them when they are published. Such systems are usually supported by a middleware in charge of keeping track of subscriptions and dispatching the published messages. For scalability reasons, such middleware can be composed of multiple interconnected nodes called brokers. These brokers are organized in some overlay and collaborate with their broker neighbors to guarantee that all published messages are eventually received by their subscribers. Usually the broker overlay is fixed once for all. In this work we propose a heuristic-based strategy to modify such overlay for optimizing the message flow. The approach we adopt is inspired by self-organization in biology and makes the broker overlay behave as an autonomic distributed system. We formally model the approach in the case of a topic-based publish-subscribe system, and prove that it is able to reduce the overall network traffic even in the presence of uncertainty and churn among the brokers. We also generalize the approach in the case of content-based publish-subscribe systems with the support of extensive simulation data.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

Keywords

Self-organization, publish-subscribe systems, adaptive routing, autonomic computing, bio-inspired algorithms

ICAC-09, June 15-19, 2009, Barcelona, Spain.

1. INTRODUCTION

The publish-subscribe paradigm has been recognized as particularly interesting in all cases in which complex distributed systems composed of highly decoupled components need to be developed [11, 16]. According to this paradigm a system is characterized by two types of components: *publishers* and *subscribers*. Publishers publish messages without knowing the recipients, subscribers subscribe to some kinds of messages and receive them when they are published. Such systems are usually supported by a middleware — whose main component is called *dispatcher* — that is in charge of keeping track of subscriptions and of dispatching the published messages to the interested parties. The main advantage offered by this approach is the loose coupling between the system components: if a component wants to send a message, it does not need to know the destination(s) (*anonymity*), while a destination does not need to know the source in order to receive that message.

However the underlined architecture has some limitations in terms of scalability and reliability as the dispatcher can be a single point of failure. To overcome this issue the idea of a distributed dispatcher has been introduced in various approaches [18]. This is composed of a network of nodes called *brokers* interconnected in some overlay network and cooperating to manage subscriptions and published messages. This way the failure of one broker does not take down the whole system, provided that some mechanisms for reconfiguring the network are defined. This distributed solution has two open problems, the first is how to efficiently send subscriptions/publications across the broker overlay, the second one is how to choose the right overlay to minimize network traffic. The first problem is addressed for instance in [10, 21, 24] while possible solutions for the second one are proposed for instance in [17, 3, 15].

In this paper we focus on the second problem and propose an innovative solution to minimize the cost of routing messages by periodically adapting the topology of the dispatcher overlay.

This approach takes some elements from bio-inspired evolutionary computing, such as the need of a minimal amount of randomness to achieve optimal results [19, 14]. Moreover it is able to self-optimize network traffic relying only on the local knowledge of each node of the network. The proposed solution has been formalized and proved for a simple *topic-based* publish-subscribe system and the same results have been validated experimentally to the more general *content-based* case. The obtained results clearly show some advan-

tages compared to the algorithms defined so far and demonstrate a very interesting application of self-organization and autonomic computing.

This paper is organized as follows. Section 2 provides some background on publish-subscribe systems. Section 3 discusses some existing approaches to the traffic-reduction problem. Section 4 describes our algorithm in the topic-based case. Section 5 extends the algorithm to the content-based case. Section 6 discusses the experimental results and, finally, Section 7 concludes the paper.

2. PUBLISH-SUBSCRIBE SYSTEMS

Publish-subscribe paradigm

Publish-Subscribe systems — or simply pub/sub systems — are distributed systems where components are connected to a dispatcher. Each component may subscribe to particular messages, remove a previous subscription, or publish a message. The dispatcher is in charge of collecting subscriptions and of forwarding published messages to components that have a matching subscription.

Topic-Based vs Content-Based

A pub/sub system may be classified in two different types depending on how subscriptions to messages are expressed. In topic-based pub/sub systems each subscription contains a topic and each message is tagged as belonging to a topic: a subscription matches a message if both belong to the same topic. In content-based pub/sub systems each subscription is a matching function that is able to filter messages based on their content: a subscription matches a message if the given matching function evaluated on the message is true.

Distributed dispatcher

For scalability and dependability reasons we focus on the case in which the dispatcher is composed of a network of interconnected nodes called brokers. The arrangement in which the nodes are connected constitutes the topology of the network. The simplest topology is the star topology in which all the nodes have only a link to a central node. Another common topology is the tree topology, which is a composition of star topologies connected in a hierarchical way. The most important properties of tree topologies are the absence of loops and the presence of a single unique path between two different nodes. Finally the most general topology is the graph topology in which nodes are connected to other nodes without any constraint, so it may contain loops and redundant paths among the nodes.

Brokers are in charge of acting as a dispatcher: they collect messages and subscriptions and forward them to other brokers and components of the network as needed. This way the failure of a single broker does not affect the whole system, moreover the traffic is spread among all of them.

Message routing

In the distributed dispatcher case there are several mechanisms to route messages from the publishing component to the subscribing components. The most used approach is the *subscription-forwarding* algorithm [11]. In this approach (limited to tree topologies) each broker keeps a routing table that associates each subscription to destinations (that may be either components or other brokers). When the broker receives a new subscription from a broker or a component, it updates its routing table and forwards the subscription to all the other brokers using a flooding mechanism. This way published messages reach only destinations that have a subscription that matches those messages. In general the tree-topology limitation can be relaxed by using protocols such as the one presented in [8] in which a graph topology is reduced to the simper tree-topology case by creating multiple trees.

Overlay traffic

Each broker is aware of the traversing messages, in particular it knows for each message the previous hop, the next hop(s), and the subscription(s) that matches it. Part of this information is used in existing approaches (see Section 3) as well as in the approach we propose to perform decisions on possible overlay topology modifications to reduce the traffic. In this work the notion of traffic is defined as the total number of messages exchanged among all the brokers of the dispatching network.

3. CURRENT APPROACHES TO OVERLAY SELF-ORGANIZATION

In the literature some approaches that reduce the number of routed messages by self-organizing the overlay topology in pub/sub systems are presented.

In the topic-based pub/sub TERA [2] the authors propose an approach in which brokers subscribed to the same topics are clustered into groups, thus increasing the probability of keeping the number of brokers involved in delivering a message small. The limitation of this approach is that it requires a different overlay and cluster for each topic, therefore it is not extendible to the more generalized content-based case.

Another work from Baldoni et al. [3] tries to self-organize the broker overlay in pub/sub systems that use tree-based event routing protocols (such us REDS [12], SIENA [7]): it defines an interests similarity function between two brokers and connects together brokers that show a higher value for that function in an iterative way. One issue of this algorithm is the fact that an iteration may involve an entire path of nodes in terms of hop sequence, thus requiring a high number of messages and complex locking mechanisms; another issue is the fact that this method, in order to maintain a tree topology, may remove a link that is already part of an existing optimal path.

Similarly a recent work from Jaeger [15] proposes a self-organizing algorithm that assigns to each node a cache that stores information about the messages consumed by the broker, then the cache is filtered and, using a heuristic based on that filter and other cost functions, it takes the decision whether to rewire or not. The problem with this heuristic is that it does not assure that the rewiring would effectively reduce the cost at the global level, since a rewiring operation reduces the cost function in the local node, but, at the same time, it may increase it in a remote node. To overcome this problem the rewiring broker performs a consensus phase where it asks other brokers in its neighborhood about the increase in cost deriving from the reconfiguration, however this has the disadvantage of relying on neighborhood multicast communication very often.

There are several other approaches such as [20, 4] that can be also applied to rooted tree overlays only, but each node

needs to maintain the status of all its neighbors.

A completely different approach [25] tries to improve scalability by using unstructured overlays in which any node of the network may become the neighbor of another node, and messages are sent using flooding or epidemic protocols: the main idea is that each broker of the system tries to cluster nodes with the same interests. Strengths of this approach are the fact that it is not only scalable, but also resistant to failures and churn without extra effort. The drawback is that, unlike a full content-based system, it support only subscriptions that can be expressed as a discrete numeric value or an interval. Another epidemic-based approach has been proposed by Guerraoui et al [1]. However it is designed to work only in topic-based systems.

In the context of peer-to-peer networks (Pastry [22], Chord [23]) Castro et al. [9] propose a self-organization algorithm heuristic that is able to globally reduce a generic distance function (that may be also the communication cost). However this approach does not consider application-level information but the underlying network information only.

In the Optimal Content-Based Routing (OCBR [17]) approach, defined for systems that use the subscription-forwarding routing algorithm, each broker considers the amount of traffic that has been forwarded in the past between every couple of its neighbors: at each iteration each broker creates a direct link between the two neighbors that have the highest forwarded traffic and then disconnects itself from one of them to preserve the global number of links. This approach has the drawback of not considering information about node interests, thus it tends to perform more reconfigurations than needed.

The approach we present in this paper is similar to OCBR from the point of view of network assumptions and routing algorithm, but it considers also the relationship between traffic and subscriptions to achieve the same goal with less reconfigurations.

4. OVERLAY SELF-ORGANIZATION IN THE TOPIC-BASED CASE

In this section we introduce a heuristic to decide how to self-organize the dispatcher overlay network with the aim of reducing the communication costs of the system in terms of overlay traffic. We propose an algorithm that, using only simple local rules based on this heuristic, brings the overlay network to a more optimized configuration.

4.1 Algorithm

The approach we have adopted is composed of two phases: a *training phase* in which each broker of the network learns how many times a subscription is matched by every forwarded messages, and a *reconfiguration phase* in which the training information is used to take decisions about possible overlay reconfigurations assuming that new traffic is similar to the previous one. These overlay reconfigurations, called rewirings, consist in adding/removing neighbors to/from each broker. Since nodes in the network may be in different phases at the same time, and since there can be conflicts among concurrent reconfigurations, a locking mechanism has been introduced. The sequence diagram of the algorithm is shown in Figure 1.

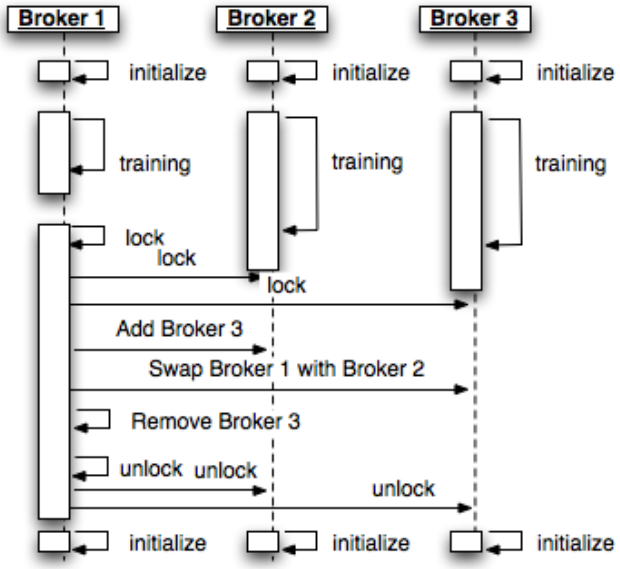


Figure 1: Algorithm Sequence Diagram

Training phase

This phase consists in creating in each node of the network a traffic counter for each pair <*source neighbor, matching-subscription*> for incoming messages, and a traffic counter for each pair <*destination neighbor, matching-subscription*> for outgoing messages. These counters are initialized to zero and their value is incremented by one as soon as an input message matching a given subscription enters the node from a source node (previous hop of the message), and after the message is forwarded to an external destination node (next hop of the message). The duration of the training phase (Δt) depends on the system traffic: shorter training means a more reactive network, but leads to a possible loss of information; longer training leads to a more accurate information, but the reconfigurations are delayed and less reactive. The right duration can be set using a time threshold or a message threshold: the best value is application dependent.

Reconfiguration Phase

In this phase every node calculates, for each pair of neighbors, the amount of messages that would be saved after a topology reconfiguration involving them (a formal definition of reconfiguration is presented in Section 4.3). For a generic reconfiguration R this number can be calculated using the utility function $U(R)$ that will be formalized in Section 4.4: this function gives an estimate on the number of messages that would be saved after applying the reconfiguration R. After evaluating $U(R)$ for every possible reconfiguration R, if there is at least a positive value for $U(R)$ we choose to perform a rewiring using the two neighbors of the local node that maximize $U(R)$. To prevent synchronization issues all the nodes involved in R are locked before applying the reconfiguration, then, after it has been applied, nodes are unlocked, all their counters are reset to zero, and a new training phase begins.

The following sub-sections describe how we compute the utility function used in the reconfiguration phase. We start defining the model of a pub/sub system (Section 4.2) and of

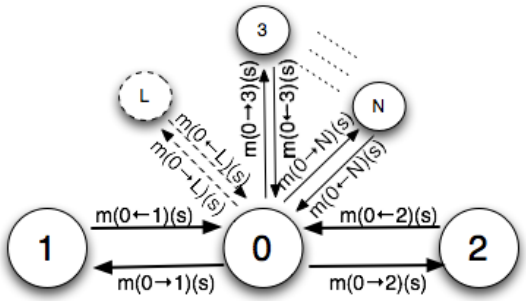


Figure 2: Neighborhood Model

a possible reconfiguration (Section 4.3). Finally, we calculate the utility function in terms of the messages that would be saved thanks to the reconfiguration (Section 4.4).

4.2 System Model

The pub/sub system is modeled by a non-directed bipartite graph $G = (C, B, E, D)$ where C is a set of vertices called component nodes, B is a set of vertices called broker nodes, E is a set of edges that connect components to brokers, D is a set of edges that connect brokers to other brokers. Each component C_i at a certain time t has a list of messages to be published M_i and a list of issued subscriptions S_i. Messages are in general unconstrained and can contain arbitrary data, while subscriptions are conditions over the messages: a subscription matches a message if its condition evaluated on the message is satisfied (content-based case). In the topic-based case messages belong to a class and subscriptions select messages corresponding to the same class. In this model the subscriptions are considered constant since we refer to a snapshot of the network.

The following is a list of information that is locally available to a generic broker node $x \in B$, assuming that the routing of messages in the system can be treated using the subscription-forwarding approach described in Section 2:

- *Ne(x)*: subset of B containing brokers that are directly connected to x, also known as neighborhood, plus a virtual node L that is used to model all the components directly connected to x. This virtual node is the source/drain for the traffic generated by local components and it has been introduced to keep the analytical expressions as simple as possible.
- S: set of all subscriptions (this information is the same in all the brokers of the network since we are assuming that each time a new subscription is issued it is forwarded to all the brokers of the network with an atomic operation).

Another information that can be directly measured by the generic broker x in a time interval Δt is the number of messages that are sent from itself to a generic neighbor $y \in Ne(x)$ and vice-versa that match a generic subscription $s \in S$. We refer to these measurements as traffic counters and the notation $m^{(s)}_{x \to y}$ for outgoing messages, and $m^{(s)}_{x \leftarrow y}$ for incoming messages.

In Figure 2 we can see how the traffic counters are associated to links. Circles represent brokers: *0* is our reference broker, *1..N* are the neighbors brokers, and L is the virtual neighbor broker that represents local components. Arrows represent the links among brokers and their labels are the traffic counters measured by node *0*.

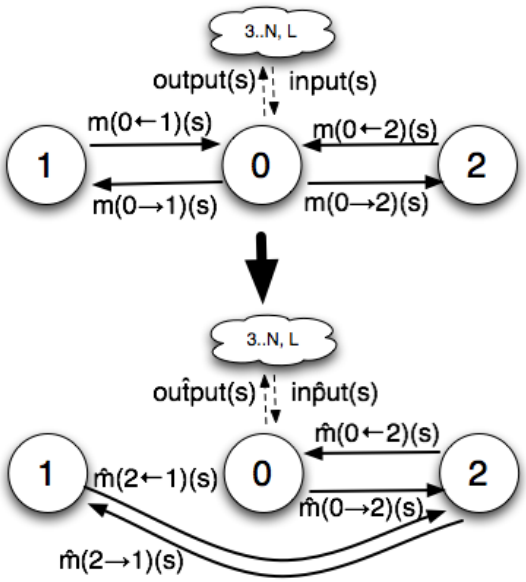


Figure 3: Links before and after a reconfiguration started on node N_0.

4.3 Reconfiguration Model

Definition Reconfiguration

Given a generic broker $N_0 \in B$ called the local broker, and two different non-virtual neighbors $N_1, N_2 \in Ne(N_0)$ such that $N_1 \notin Ne(N_2)$. A reconfiguration (or rewiring) R=<N_0,N_1,N_2> is defined as the removal of the bidirectional link between N_0 and N_1, and the insertion of a bidirectional link between N_1 and N_2.

In other words a reconfiguration R=<N_0,N_1,N_2> moves broker N_1 from the neighborhood of N_0 to the neighborhood of N_2. Figure 3 shows how the local view of node N_0 looks like before and after the reconfiguration R. Neighbors that are not involved in the reconfiguration are grouped together and depicted as a cloud, and all the links to these nodes are grouped in a single bidirectional link labeled with the traffic counters $input^{(s)}_{N_0}$ and $output^{(s)}_{N_0}$. From now on we refer to these brokers as *other brokers,* and the incoming/outgoing traffic counters can be calculated as the sum of the traffic of the grouped links:

$$input^{(s)}_{N_0} = \sum_{\substack{N_j \in Ne(N_0) \\ N_1 \neq N_j \neq N_2}} m^{(s)}_{N_0 \leftarrow N_j}$$

$$output^{(s)}_{N_0} = \sum_{\substack{N_j \in Ne(N_0) \\ N_1 \neq N_j \neq N_2}} m^{(s)}_{N_0 \to N_j}$$

The traffic counters after the reconfiguration are depicted in Figure 3 as $\hat{m}$. Their value represents the amount of

traffic that would have traversed the corresponding link if the reconfiguration had occurred before receiving the traffic. In other words these counters are estimators of future traffic in case future traffic is similar to the previous one, which is usually a reasonable assumption in pub/sub systems.

THEOREM 1. *Given that $G = (C, B, E, D)$ is a network of brokers representing a distributed dispatcher of a topic-based pub/sub system that uses the subscription-forwarding routing approach, $N_0 \in B$ is a generic broker called the local broker, S is a subscriptions set, R=<N_0,N_1,N_2> is a valid reconfiguration, and $m^{(s)}_{N_0 \leftarrow N_j}$ and $m^{(s)}_{N_0 \rightarrow N_j}$ are the subscription-specific traffic counters measured by N_0 for a generic neighbor N_j over the time interval Δt.*

The traffic counters after the reconfiguration R are expressed by the following relations:

$$\hat{m}^{(s)}_{N_2 \leftarrow N_1} = m^{(s)}_{N_0 \leftarrow N_1} \tag{1}$$

$$\hat{m}^{(s)}_{N_2 \rightarrow N_1} = m^{(s)}_{N_0 \rightarrow N_1} \tag{2}$$

$$\hat{m}^{(s)}_{N_0 \leftarrow N_2} = \begin{cases} m^{(s)}_{N_0 \leftarrow N_1} + m^{(s)}_{N_0 \leftarrow N_2} & if\ output^{(s)}_{N_0} > 0 \\ 0 & otherwise \end{cases} \tag{3}$$

$$\hat{m}^{(s)}_{N_0 \rightarrow N_2} = \begin{cases} input^{(s)}_{N_0} & if\ m^{(s)}_{N_0 \rightarrow N_1} > 0 \vee m^{(s)}_{N_0 \rightarrow N_2} > 0 \\ 0 & otherwise \end{cases} \tag{4}$$

$$\hat{input}^{(s)}_{N_0} = input^{(s)}_{N_0} \tag{5}$$

$$\hat{output}^{(s)}_{N_0} = output^{(s)}_{N_0} \tag{6}$$

PROOF. Relation 1 is explained by the fact that the remote nodes of N_1 remain the same, thus the joint subscriptions of those nodes is equal to the situation before the reconfiguration: the result is that N_1 sends the same messages.

Similarly Relation 2 is explained by the fact that the subscriptions of N_1 are constant, therefore, since in the subscription-forwarding approach a node receives only messages it is interested to, the messages it receives from its new neighbor N_2 are the same he would have received from N_0.

The explanation of Relation 3 is given by the fact that after the reconfiguration N_0 receives from N_2 the messages generated by N_2 ($m^{(s)}_{N_0 \leftarrow N_2}$) plus the messages generated by N_1 that were previously sent to it directly ($m^{(s)}_{N_0 \leftarrow N_1}$). Having a value of zero for $output^{(s)}_{N_0} = 0$ means that neither N_1 nor N_2 are subscribed to s, therefore there is no need to forward messages from N_2 to N_0 and the corresponding counter can considered zero. This situation happens when N_1 sends messages that match a topic subscribed by N_2 that has not been subscribed by the other nodes.

Relation 4 is proved by considering the outgoing messages sent from N_0 to N_1 and N_2 before the reconfiguration. If there were no outgoing messages before the reconfiguration, then there would not be outgoing messages also after the reconfiguration, since the joint subscriptions of N_1 and N_2 would not match any message too. In the case there are outgoing messages to N_1 or N_2 before the reconfiguration, then these messages would necessarily come from the other brokers, thus the traffic sent from N_0 to N_2 after the reconfiguration is equal to $input^{(s)}_{N_0}$.

Finally $input^{(s)}_{N_0}$ and $output^{(s)}_{N_0}$ (Relations 5 and 6) before and after the reconfiguration remain constant because the corresponding nodes were not involved in the reconfiguration and therefore their traffic patterns do not change.

□

4.4 Traffic Reduction Utility Function

The results of Theorem 1 can be used to compare the traffic before and after the reconfiguration R=<N_0,N_1,N_2>. In particular the number of messages transferred before and after the reconfiguration are expressed the following way:

$$MsgsPreReconf_R = \sum_{s \in S}(input^{(s)}_{N_0} + output^{(s)}_{N_0} + \\ + m^{(s)}_{N_0 \leftarrow N_1} + m^{(s)}_{N_0 \rightarrow N_1} + m^{(s)}_{N_0 \leftarrow N_2} + m^{(s)}_{N_0 \rightarrow N_2}) \tag{7}$$

$$MsgsPostReconf = \sum_{s \in S}(\hat{input}^{(s)}_{N_0} + \hat{output}^{(s)}_{N_0} + \\ + m^{(s)}_{N_0 \leftarrow N_1} + m^{(s)}_{N_0 \rightarrow N_1} + \hat{m}^{(s)}_{N_0 \leftarrow N_2} + \hat{m}^{(s)}_{N_0 \rightarrow N_2}) \tag{8}$$

Thus, after substituting the relations obtained so far, the total number of messages that is saved by the reconfiguration is the following:

$$U(R) = MsgsPreReconf_R - MsgsPostReconf_R = \\ = \sum_{s \in S}(\hat{m}^{(s)}_{N_0 \leftarrow N_2} + \hat{m}^{(s)}_{N_0 \rightarrow N_2} - m^{(s)}_{N_0 \leftarrow N_2} - m^{(s)}_{N_0 \rightarrow N_2}) \tag{9}$$

This relation is the utility function *U(R)* that we evaluate in the reconfiguration phase of the algorithm.

5. OVERLAY SELF-ORGANIZATION IN THE CONTENT-BASED CASE

We can apply to the content-based case the reconfiguration approach that we have presented in Section 4 keeping in mind that due to the possibility of having partially overlapping subscriptions, Theorem 1 is no longer valid since a single message can match more than a single subscription. In fact, even if the value obtained by the utility function U is no longer the actual number of saved messages, it can still be used as an approximate traffic indicator: as this number becomes higher, the higher is the probability of reducing the traffic. Of course this calculation can lead to an overestimation of the result, but the simulation phase (see Section 6) revealed that it is non-dominant, therefore the utility function is still good for moving towards more optimized overlay configurations.

5.1 Non-deterministic approach with randomness.

From the performed simulation experiments we have observed that the simple non-deterministic approach tends to

reduce the number of messages, but also provides a solution that is often suboptimal when compared to other existing techniques like the OCBR approach we have presented in Section 3 (see Section 6 for details about performance). This can be explained by the fact that our utility function tends with a higher probability to underestimate the so-called saved traffic indicator, therefore most of the candidate nodes are not considered good enough to be rewired. The result is that the algorithm is over-constrained and, after a faster convergence, it gets easily stuck in local optima. A possible solution to this problem (similarly to evolutive/genetic algorithms) is to add a perturbation to the utility function in the form of randomness or noise. In the previous approach we have chosen to rewire only if the utility function is positive, but if we consider an acceptance threshold that is lower than zero we are increasing the number of "noisy" iterations since we perform rewirings that have a higher probability of increasing the network traffic. With this method the traffic may be temporary increased but this way it is possible to escape from local maxima and obtain a plateau that is comparable to OCBR, but with less overall node rewirings.

5.2 Generalization

In a real network the links among nodes are not necessarily equal to each other: they may have different capacities, different latencies, and different usage costs. This way the metric we want to optimize cannot be simply related to the number of the messages as we have assumed until now, but it is a more complex weighted function of each link that may change over time. Our approach may be generalized by using an extended utility function which multiplies the messages counters of each link by that weighted function, but we will not go fully into detail as it is application dependent.

6. EXPERIMENTS

In this section we evaluate the performance of the algorithms in different situations in order to verify experimentally the theoretical results. In all the situations we will compare our results to the ones obtained using OCBR algorithm [17]. The choice of OCBR is due the fact that, as we have discussed in Section 3, this algorithm, like the one we propose, is completely decentralized, scalable and with low requirements on node knowledge and interactions.

6.1 Tests and Testbed

Methodology

The simulations have been carried out using a custom simulator written in Java and described in [13]. The peculiarity of this simulator is that its results are not biased by the policy that is used to forward the component subscriptions and by the transport method that is used to transfer the messages because message forwarding and overlay rewirings occurs in independent simulation steps that do not influence each other. A simulation is run at least 20 times to provide some statistical validity to the obtained results. Every simulation is composed by looping iterations that consist of the following sequential phases:

1. *Subscriptions initialization*: random subscriptions are generated and forwarded to all the nodes.
2. *Messages generation*: random messages are generated and published by some of the nodes.
3. *Training*: an overlay is created according to the specified topology, then messages are forwarded to the destination nodes: every time a message traverses a node, it modifies its traffic counters according to the algorithm that we want to evaluate.
4. *Reconfiguration*: some brokers are rewired according to the algorithm used.

Simulation parameters

The simulation parameters that have been considered are related to the characteristics of the environment on which a distributed pub/sub middleware is deployed. Unfortunately at the moment there are no real traffic logs or dumps that can be used to perform a more realistic simulation, therefore we have made the same assumptions that we have found in some works of the literature such as [17].
The customizable simulation parameters are the following:

- *Number of brokers*: total number of brokers.
- *Number of components*: number of components that are connected to each broker. We have chosen to have only one component for each broker since more components can always be modeled as a single one with joint interests.
- *Broker topology*: this parameter defines the initial structure of the overlay network. With this parameter we need to create an initial pattern of interconnections that is similar to what we can find in real networks. The topologies we have considered in this work are: tree-based scale-free topologies in which the probability of a broker to be connected to another broker is $P(k) = k^{-\lambda}$ where λ is a generic constant between 2 and 3 for most real networks [5]; spiral topologies in which two extreme nodes have a single neighbor and all the other intermediate nodes have exactly two neighbors.
- *Percentage of subscribers*: percentage of brokers that may receive subscriptions from their component.
- *Percentage of publishers*: percentage of brokers that may receive publications of messages from their component.
- *Number of subscriptions per subscriber*: this parameter indicates how many subscriptions are sent by each subscriber. As this value gets higher, the traffic and the chance of a node to match a generic message increases.
- *Number of publications per publisher*: this parameter controls the traffic generated on the network. Higher values for this parameter consist in a longer training phase in the simulation, lower values instead may inhibit the algorithm for lack of traffic information.
- *Number of different topics*: this value represents the total number of available different topics.

Table 1: Most significative reference experiments with some variations on the simulation parameters. TB means Topic-Based, CB means Content-Based.

Parameters	Ref. experiment	Variations
Number of brokers	500	-
% Subscribers	100%	20%, 50%
% Publishers	20%	50%
Publications per publisher	150	50
Subscriptions per subscriber	3	-
Initial overlay topology	Scale-Free	Spiral
Filter size	1 (TB), 3 (CB)	-
Number of types of messages	10	-
Perturbations and/or churn	No	Yes
Utility Function Threshold	0 (TB), 10 (CB)	0 (CB), 50 (CB)

- *Cardinality of composed subscriptions*: this value indicates the presence of composed subscriptions, where a composed subscription is a subscription that may match more than a topic. In topic-based routing experiments it is defined as one since there are no composed topics, while in the content-based case its value may be larger than one. This choice is explained by the fact that a topic-based middleware in which a single subscription may match multiple topics has the same expressive power of a content-based middleware. In this study we consider composed subscriptions of 3 topics for the content-based case.

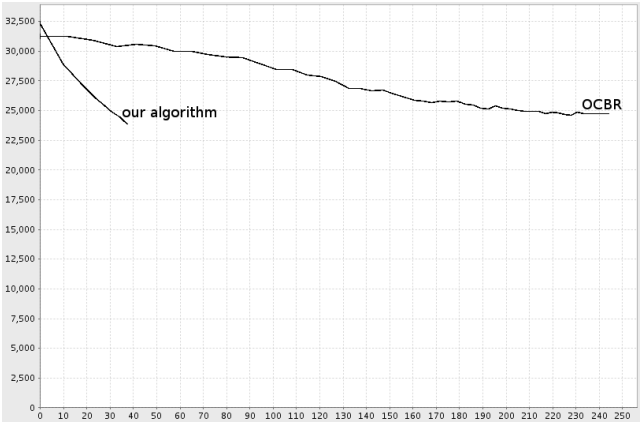
- *Distribution of messages*: this is the statistic distribution function that is used to model the belonging of a message to a given topic. We have used a uniform distribution.

- *Distribution of subscriptions*: this is the statistic distribution function that is used to model the belonging of a subscription to one (or more) given topic. The distribution adopted is the *Zipf* [6] with parameter 1 since it is the same used in the work we are using to compare our results [17].

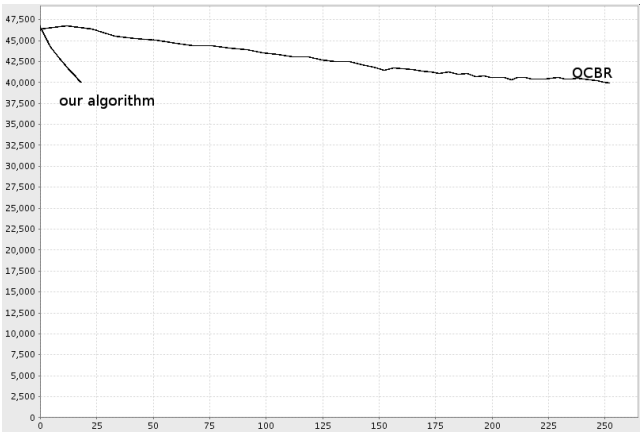
- *Utility Function Threshold*: this is the threshold T that we use to decide if the reconfiguration R should be applied. R is applied if $U(R) > T$. Since $U(R)$ has positive values for reconfigurations R that have a high probability to reduce traffic and negative values for reconfigurations that increase traffic we may use negative values for T to add randomness to the algorithm. The values we have considered for T are: 0 to simulate the approach without randomness; -10 and -50 to simulate the approach with two different levels of randomness.

Experiments

The experiments have been executed in the following way: first of all we have performed several experiments to see if the algorithm was really able to reduce the network traffic using different simulation parameters, then we have selected the most significative experiments (called from now on *reference experiments*) and we varied one parameter at the time to see their impact on the algorithm results.



(a) topic-based case



(b) content-based case

Figure 4: Reference experiments. Comparison between our algorithm and OCBR. x-axis contains the number of rewirings, y-axis contains the number of exchanged messages.

Table 1 shows the most significant reference experiments in the topic-based and in the content-based setting with variations of their input parameters. Each value of the table has been used in both settings with the exception for the filter size and the utility function threshold that are fixed respectively to 1 and 0 in the topic-based case.

6.2 Results

Reference experiments

In this paragraph we will show some of the most interesting results that have been obtained from the simulations. A more detailed set of charts and simulation data can be found in a technical report [13]. The simulations that have been carried out have applied our overlay self-organization algorithm to both topic-based (without randomness) and content-based (with randomness) systems to reduce the total number of messages. In our notion of traffic we do not consider the size of the message and if a message traverses N brokers (hop nodes) before going from the publisher to all the correct subscribers, it counts as N messages.

The chart in Figure 4a shows the reference experiments of

our algorithm and OCBR in the topic-based settings. The lines represent the average values of the network traffic. Figure 4b shows the same experiments run in the content-based setting. The size of the interval between the minimum and the maximum value of the traffic in our set of experiments tends to be constant with a difference of +/-10%. This difference is due to the fact that each simulation is run with a different seed, therefore the initial topology, the publications and the subscriptions will be slightly different. However the interval is small enough to consider the data statistically valid. From the comparisons in Figure 4 we can see that the heuristic of our algorithm is able to reduce the traffic with a much lower number of rewirings than the other approach in the same amount of time steps, this explains also why in the chart the line of OCBR is longer than the other.

Subscribers and publishers variations

Figures 5ab and 6ab show how the convergence rate of the algorithms changes in presence of variations of the number of subscribers and of the publishers. We can see that the shape of the curves is similar to the reference experiment. All the lines show a different initial/final number of exchanged messages: this difference may be explained by the fact that in the different considered situations we have a different number of sources (publishers)/recipients (subscribers) nodes.

Initial Topology Variations

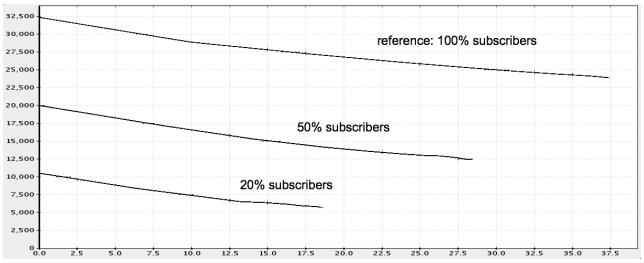
Figures 5c and 6c show that the initial topology has an impact on the initial iterations of the algorithm since the spiral configuration is the worst possible configuration due to its highest network diameter. Looking at the values obtained in the last simulation steps (when the configuration becomes stable) we can see similar values with both topologies: this shows that the dominant attractor brings the overlay towards a unique optimized topology that does not depend in the long runs on the initial one.

Perturbations and/or churn

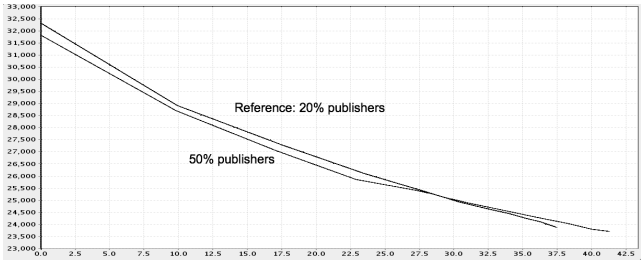
In this experiment we added some perturbations to the network like varying the nodes or the subscriptions/publications between different iterations of the same simulation. This test has been done to see if the algorithm has overfitting problems and if it is reliable in presence of uncertainty. From Figures 5d and 6d we can notice a shape that looks very similar to the one without perturbations, therefore our algorithm shows a good level of robustness since it is still able to self-optimize. This can be explained also by the fact that perturbations are a form of noise/randomness that in this type of situations it is not usually a problem, but it also prevents local optima and improves the effectiveness of the algorithm.

Utility Function Threshold Variations

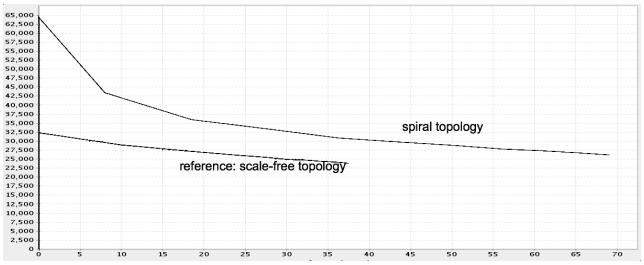
This experiment has been run in the content-based setting only to investigate the effects of artificially added randomness to the algorithm execution. The method we used to add or modify the randomness is to change the threshold for which the utility function *U(R)* defined in Section 4.4 triggers a reconfiguration. As we expected from the previous theoretical considerations, Figure 7 shows the lowest traffic reduction in the case with lowest noise (utility function $U(R)>0$) because the system gets stuck in local optima, and a slower convergence in the case with highest noise (utility



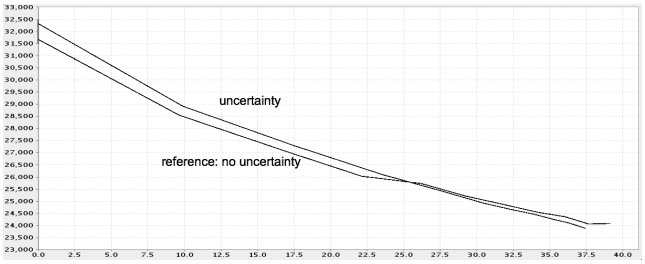
(a) Varying subscribers number.



(b) Varying publishers number.

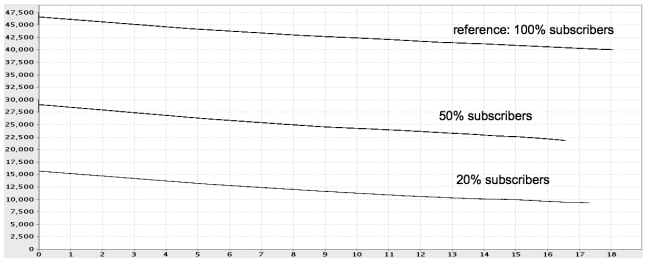


(c) Varying initial topology.

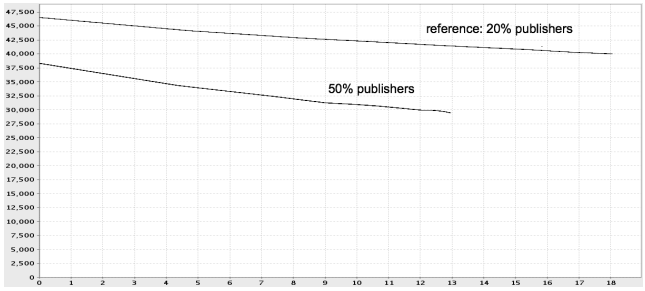


(d) Adding perturbations.

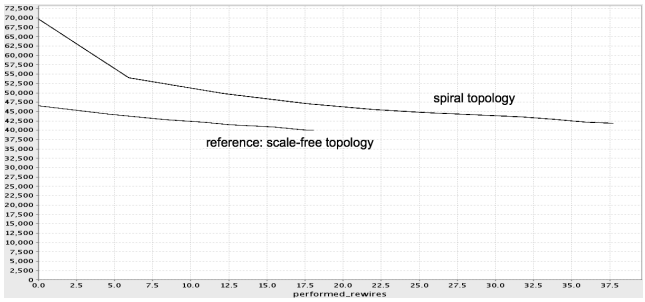
Figure 5: Variations on parameters. x-axis contains the number of rewirings, y-axis contains the number of exchanged messages. Topic-based case.



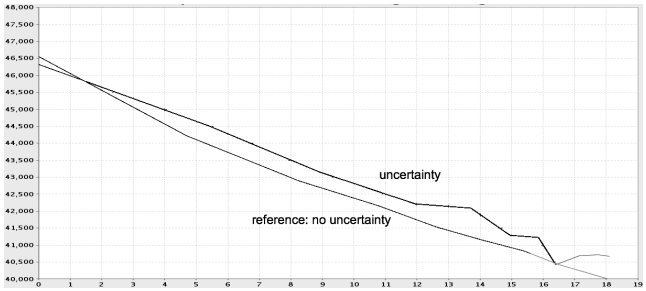
(a) Varying subscribers number.



(b) Varying publishers number.



(c) Varying initial topology.



(d) Adding perturbations.

Figure 6: Variations on parameters. x-axis contains the number of rewirings, y-axis contains the number of exchanged messages. Content-based case.

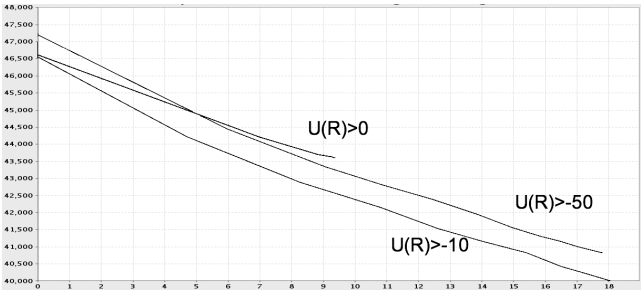


Figure 7: Effects of randomness to improve our algorithm in the content-based setting. x-axis contains the number of rewirings, y-axis contains the number of exchanged messages.

function $U(R)$>-50) because the system performs too many iterations that are not able to reduce the traffic in a significative way. The best solution is to have a tradeoff between convergence rate (high with a lower randomness) and effectiveness in terms of saved messages (high with a higher randomness).

6.3 Discussion

The results show that in all the considered situations both algorithms (our algorithm and OCBR) are able to reduce the total number of messages in a certain amount of time, but using our approach we have a faster convergence with less overhead in terms of performed rewirings. In the OCBR approach algorithm counters are updated on the basis of previous-hop and next-hop couple of nodes, while our approach needs to consider for each message also the matching subscription. This adds to each counter update operation the cost of the execution of the matching function, however this issue is not critical since usually the matching function is executed anyway to identify where to route the message. Clearly in our approach the choice for the right amount of randomness and the duration of the training phase are the keys to obtain good performance; choosing bad parameters may led the algorithm behavior to results that are worse than OCBR and therefore they should be carefully chosen before deploying the system. In conclusion in these experiments we have validated the theoretical results of Sections 4 and 5, in particular we have seen how the algorithm performance is affected by some changes in the running environment and the actual benefits that we have by adding artificial randomness to the algorithm.

7. CONCLUSIONS

In this paper, we propose, analyze and prove a heuristic to organize the broker overlay for self-optimizing the number of messages that are exchanged in a multi-broker topic-based and content-based publish-subscribe system. The difference with respect to competing approaches is that the amount of overlay reconfigurations in our case is sensibly smaller when compared to the other cases. Therefore the algorithm overhead in terms of exchanged messages is smaller.

Finally we have seen that adding a certain amount of randomness to the algorithm plays a key role in obtaining a good solution.

Some of the future challenges in this field are to dynamically adapt this level of randomness to keep into account the cost caused by the algorithm used to update the subscription tables after a reconfiguration, and to find a heuristic to self-determine the duration and the rate of each algorithm phase.

8. ACKNOWLEDGEMENTS

This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227077-SMScom.

9. REFERENCES

[1] S. Baehni, P. T. Eugster, and R. Guerraoui. Data-aware multicast. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 233, Washington, DC, USA, 2004. IEEE Computer Society.

[2] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 2–13, New York, NY, USA, 2007. ACM.

[3] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. *Comput. J.*, 50(4):444–459, 2007.

[4] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 2:1521–1531 vol.2, March-3 April 2003.

[5] A.-L. Barabasi and A. Reka. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. pages 126–134, 1999.

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19:332–383, 2001.

[8] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *IEEE INFOCOM '04*, 2004.

[9] M. Castro, P. Druschel, Y. Charlie, and H. A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical report, 2002.

[10] G. Cugola, D. Frey, A. L. Murphy, and G. P. Picco. Minimizing the reconfiguration overhead in content-based publish-subscribe. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1134–1140, New York, NY, USA, 2004. ACM.

[11] G. Cugola, E. D. Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Transactions on Software Engineering*, 27(9):827–850, 2001.

[12] G. Cugola and G. P. Picco. Reds: a reconfigurable dispatching system. In *SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware*, pages 9–16, New York, NY, USA, 2006. ACM.

[13] D. Dubois. Overlay Self-Organization for Traffic Reduction in Multi-Broker Publish-Subscribe Systems. Technical report, Politecnico di Milano, 2008. http://home.dei.polimi.it/dubois/techRepPS08.pdf.

[14] J. Deneubourg and al. Probabilistic behaviour in ants: a strategy of errors? *J. of Theoretical Biology*, pages 105, 259–271, 1983.

[15] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann. Self-organizing broker topologies for publish/subscribe systems. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 543–550, New York, NY, USA, 2007. ACM.

[16] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[17] M. Migliavacca and G. Cugola. Adapting publish-subscribe routing to traffic demands. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 91–96, New York, NY, USA, 2007. ACM.

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

[19] S. Nicolis and al. Optimality of collective choices: a stochastic approach. *Bulletin of Mathematical Biology*, pages 65, 795–808, 2003.

[20] O. Papaemmanouil, Y. Ahmad, U. Çetintemel, J. Jannotti, and Y. Yildirim. Extensible optimization in overlay dissemination trees. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 611–622, New York, NY, USA, 2006. ACM.

[21] H. Parzyjegla, G. G. Muhl, and M. A. Jaeger. Reconfiguring publish/subscribe overlay topologies. *Distributed Computing Systems Workshops, International Conference on*, 0:29, 2006.

[22] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Lecture Notes in Computer Science*, pages 329–350, 2001.

[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[24] P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. *Distributed Computing Systems, International Conference on*, 0:562–571, 2004.

[25] S. Voulgaris, E. Riviere, A. marie Kermarrec, and M. V. Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *In IPTPS '06: the fifth International Workshop on Peer-to-Peer Systems*, 2006.