Self-Optimizing Hybrid Routing in Publish/Subscribe Systems

Arnd Schröter Daniel Graff Gero Mühl
Jan Richling Helge Parzyjegla

Communication and Operating Systems Group (KBS),
Berlin University of Technology, Germany
{aschroet,dgraff,gmuehl,richling,parzy}@cs.tu-berlin.de

Abstract. Rendering networks and distributed systems self-managing and self-optimizing has become a major research focus. This task is especially important for systems, such as publish/subscribe systems, that are used in dynamic environments. In these settings, a static configuration usually leads to a largely suboptimal system performance, while manual optimization is either too expensive or not possible at all. Distributed publish/subscribe system are usually realized by a broker overlay network providing the functionality of a decentralized notification service. In this paper, we present an approach that self-optimizes the routing configuration of a publish/subscribe broker overlay network to minimize the operational costs of the system without any manual intervention. It is based on a new class of routing algorithms, which allows a link-by-link adaptation of the routing configuration at runtime. Based on a local decision criterion, a decentralized optimization algorithm is introduced that generates only marginal extra traffic.

Keywords: Publish/Subscribe Middleware, Self-Optimization

1 Introduction

Today, many administrative tasks in networks and distributed systems are still executed manually by human operators. However, due to the increasing complexity and dynamics of these systems, this cost-extensive approach comes to a limit. Because of this, a major research focus of the recent years was to render these systems self-managing such that they can, for example, self-optimize their configuration to maximize the system's performance without human intervention.

Publish/subscribe systems provide an asynchronous, anonymous, and data-centric many-to-many communication model and decouple the communication partners in time, space, and flow [5]. Leveraging this flexibility, publish/subscribe infrastructures are increasingly often used to build novel distributed systems and applications ranging from information and content dissemination over distributed event processing to system integration and monitoring. Publish/subscribe systems are especially interesting in the context of self-managing systems

for two main reasons: First, they are an ideal basis for realizing autonomic and self-managing distributed systems and, second, because of their complexity they should be made self-managing, too.

Components within a publish/subscribe system communicate by producing and consuming notifications. While *producers* aka *publishers* publish notifications, *consumers* aka *subscribers* subscribe for notifications they are interested in (e.g., using content-based filtering). The publish/subscribe system acts as *notification service*, which connects the components and delivers published notifications to all consumers with matching subscriptions. Publish/subscribe middleware systems (e.g., HERMES [9] and REBECA [6]) are usually implemented as a set of cooperating brokers each managing a set of *local clients*. The brokers are connected by *overlay links*, which are used to dispatch published notifications as well as issued and revoked subscriptions. For this purpose, each broker manages a *routing table*, which is used to forward incoming notifications to interested neighbor brokers and local clients. The employed routing algorithm determines the strategy by which routing tables are updated (e.g., in case of newly issued or canceled subscriptions) and has a major influence on the system's efficiency. There is a general trade-off between the traffic needed by the routing algorithm to keep its tables up-to-date and the number of superfluous notifications that are forwarded unnecessarily when, otherwise, all notifications are simply flooded in the network. Thus, depending on the client's distribution and the dynamics of their interests and publication behavior, different routing algorithms are better suited in different situations. However, current publish/subscribe middleware implementations usually enforce the usage of a single, statically configured algorithm, which is employed system wide. Obviously, this leads to suboptimal results as distributions, interests, and publications may change over time and may also substantially differ in distinct parts of the network.

To overcome these limitations, we investigate hybrid routing configurations, which allow the usage of more than one routing algorithm within the network. This paper extends our previous work [10], presents a new formalism and evaluates it by simulations. Specifically, we define a novel class of hybrid routing schemes by determining constraints for valid configurations, which ensure a correct delivery of all notifications. Furthermore, it is shown how valid routing configurations can be transformed into each other—seamlessly during runtime—by subsequently switching algorithms on a link-by-link basis. Leveraging this technique, a heuristic is developed, which continuously adapts the routing configuration to reduce the overall traffic. The presented algorithm is completely decentralized, requires local knowledge only, and makes the publish/subscribe system adaptive to changing distributions of clients, interests, and publications.

The rest of the paper is structured as follows: Sect. 2 discusses related work. After giving some basics in Sect. 3, we introduce the new class of hybrid routing algorithms in Sect. 4. Then, in Sect. 5 we show how the reconfiguration of the routing algorithm can be used to self-optimize the publish/subscribe system. Finally, we present an evaluation of our concept in Sect. 6 and present our conclusions in Sect. 7.

2 Related Work

Using several routing algorithms in a single publish/subscribe system was first suggested by Carzaniga [4]. He proposed to have clusters using hierarchical routing that are connected by an acyclic peer-to-peer protocol, where the structure of such a hybrid topology is derived manually from requirements in companies by the system administrators.

Rendering publish/subscribe systems self-optimizing has already been approached in the past. Bittner and Hinze [1,2,3] introduced subscription pruning, which reduces the matching costs by replacing subscription predicates by more general predicates, which are simpler to evaluate. However, this potentially leads to increased costs for forwarding notifications since more notifications may be forwarded in which no client is interested in. To decide which predicates should be pruned, each broker determines a measure called selectivity degradation.

Jaeger et al. [7] propose to adapt the interconnection of the brokers in a publish/subscribe overlay network to optimize the routing paths connecting publishers with interested subscribers. Since finding optimal overlays is NP-hard, a cost- and interest-driven heuristic only using local knowledge to determine reasonable edge substitutions is applied. The reconfigurations do not lead to service interruptions and multiple reconfigurations can be executed in parallel.

Both approaches discussed above reveal valuable insights on how to design self-optimizing publish/subscribe systems. While our approach in principle pursuits the same objective, it dynamically adapts the applied routing algorithms to achieve cost reductions instead of modifying subscriptions or changing the overlay topology. Thus, our approach can be seen as orthogonal concept which may be combined with the others.

3 Routing Algorithms in Publish/Subscribe Systems

In this section we lay the foundation for further discussions by introducing the basic network model (Sect. 3.1) and two standard routing algorithms (Sect. 3.2), which we consider in this paper.

3.1 Network Model

A notification service consists of a set of brokers $\mathcal{B}$ that are connected by an acyclic overlay. An overlay link between two brokers $B_i, B_j \in \mathcal{B}$ is described by two directed edges e_{ij} and e_{ji}, which are elements of the set of all directed edges $\mathcal{E}$. A directed edge e_{ij} indicates the flow of notifications from B_i to B_j and the reverse flow of subscriptions from B_j to B_i. As depicted in Fig. 1, we define the set of *predecessors* $\mathcal{P}_{ij}$ and the set of *successors* $\mathcal{S}_{ij}$ of a certain edge e_{ij} as follows: $\mathcal{P}_{ij} = \{e_{ki} \mid e_{ki} \in \mathcal{E} \ \wedge \ k \neq j\}$ and $\mathcal{S}_{ij} = \{e_{jk} \mid e_{jk} \in \mathcal{E} \ \wedge \ k \neq i\}$. The predecessor edges $\mathcal{P}_{ij}$ connect neighbor brokers from which notifications are potentially received before they are forwarded via e_{ij}. $\mathcal{P}_{ij}$ also describes the potential destinations of a subscription that is forwarded from B_j to B_i. On the

other hand, successor edges $\mathcal{S}_{ij}$ are potential destinations for notifications and potential sources of subscriptions of e_{ij}.

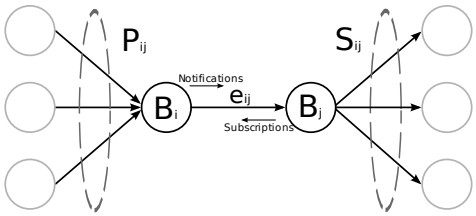

Fig. 1. Predecessors and successors of edge e_{ij}

3.2 Standard Routing Algorithms

A routing algorithm defines how routing tables are updated in reaction to issued and canceled subscriptions and how notifications are forwarded through the broker network. Typically, a single routing algorithm is applied in the whole overlay network and the applied algorithm is selected before the system is started. In this paper, we concentrate on two basic routing algorithms: *flooding* and *simple routing*. With *flooding*, no subscriptions are exchanged between brokers; instead, all notifications are flooded into the overlay such that every broker receives every notification. It is reasonable to use flooding if the majority of brokers is interested in most of the notifications; otherwise it has a overly high notification forwarding overhead. On the other hand, *simple routing* floods all subscriptions into the network such that every broker knows all subscriptions and can, thus, avoid sending notifications to neighbor brokers not needing them. Therefore, simple routing reduces the notification overhead by introducing subscription overhead. There also exist more sophisticated routing algorithms, e.g., identity-based or covering-based routing [8], which avoid sending all subscriptions everywhere by exploiting similarities among the subscriptions. To keep the presentation compact, these algorithms are not considered in this paper. However, the presented approach can be extended to cover these algorithms.

4 Hybrid Routing Algorithms

The idea of *hybrid routing algorithms* stems from the analysis of *hierarchical routing* algorithms. In hierarchical routing, there is a dedicated root broker to which all notifications and subscriptions are flooded. Downstream, notifications are filtered by active subscriptions. On the other side, subscription messages are not sent down the overlay tree. This actually means that downstream filtering algorithms, like simple routing, are used while upstream messages are flooded.

Beside hierarchical routing, there are a lot of possible combinations of routing algorithms within one overlay network. Consequentially, we now introduce the class of hybrid routing algorithms, which combines the usage of different standard routing algorithms within one broker overlay network by using different algorithms on different edges. Hybrid routing algorithms are a set of routing algorithms, whereby each individual algorithm is defined by a *configuration*, which is described in the following Sect. 4.1. Then, in Sect. 4.2 we describe how to switch from one configuration to another without service interruption.

4.1 Configuration of Hybrid Routing Algorithms

A *configuration* $R = \{r_{ij}\}$ defines which standard routing algorithm is used on which edge. The *edge configuration* $r_{ij} \in \{SR, FL\}$ defines whether flooding (FL) or simple routing (SR) is used on e_{ij}. If $r_{ij} = FL$, broker B_i forwards all notifications to B_j and B_j does not send any subscriptions to B_i. If $r_{ij} = SR$, B_j sends all subscriptions it receives from other brokers to B_i, where they are used to filter notifications that B_i is forwarding to B_j. In this way, the global nature of the standard routing algorithm is reduced to single edges.

It is not possible to set all edge configurations independently because not all configurations guarantee a correct notification service where no notifications are lost or duplicated. In the following, we derive which relation between the edge configurations r_{ij} must hold to ensure that the resulting hybrid routing algorithm works correctly. We call such a configuration a *correct routing configuration*. The crucial issue for a correct routing configuration is subscription forwarding. If one edge uses simple routing, it has to receive all necessary subscriptions for matching incoming notifications. This can be expressed by the following condition:

$$\forall(e_{i,j} \in E) : r_{ij} = SR \rightarrow \forall(e_{k,l} \in \mathcal{S}_{ij}) : r_{kl} = SR \tag{1}$$

The above equation means that the usage of simple routing on edge e_{ij} implies that all succeeding edges also use simple routing. This is necessary because the subscriptions originating from these edges are needed by broker B_i to forward all interesting notifications via e_{ij} to B_j.

Homogeneous routing algorithms (cf. Sect. 3.2) are special cases of hybrid routing algorithms, where the edge configurations are the same for all edges. Obviously, homogeneous routing algorithms satisfy the condition mentioned above. Also, for hierarchical routing this can be shown easily. Apart from these algorithms, there are many other correct hybrid routing algorithms.

4.2 Reconfiguration of Hybrid Routing Algorithms

After defining correct routing configurations, we now explain how to switch from one correct configuration to another. With this mechanism, an administrator or an algorithm for self-optimization (introduced in Sect. 5) can manage the routing scheme in the overlay network at runtime.

We denote a reconfiguration of an edge e_{ij} by $r_{ij} = [r_c, r_t]$ with $r_c \neq r_t$ and $r_c, r_t \in \{FL, SR\}$, where r_c is the current configuration and r_t the target configuration. During a *reconfiguration*, we want to guarantee the correctness of the notification service, i.e., lost and duplicated notifications must be avoided. Following, we investigate how a reconfiguration from flooding to simple routing and vice versa can be realized.

Switching from Flooding to Simple Routing. A reconfiguration from flooding to simple routing $r_{ij} = [FL, SR]$ is controlled by B_j. First, B_i must know which notifications B_j is interested in. Therefore, B_j forwards all active subscriptions that were generated by subscriptions received from successor edges $\mathcal{S}_{ij}$. Broker B_i uses these messages to build up a routing table, which represents the interests of B_j. After all active subscriptions have been forwarded, B_j sends a signal to B_i to start filtering notifications before sending them to B_j. Additionally, B_j starts to forward all subscriptions and unsubscriptions to B_i, which it is receiving from other brokers. This completes the reconfiguration to simple routing. This process implicitly assumes that Eq. 1 is satisfied. Since B_j must know the interests of its subtree for forwarding notifications to B_i, simple routing must be used downstream. In the case that B_j is a leaf broker, $\mathcal{S}_{ij}$ is an empty set and B_j can, thus, always switch its incoming edge to simple routing.

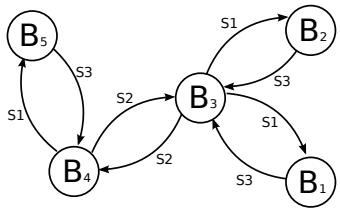

Fig. 2. Reconfiguration from Flooding to Simple Rooting

Figure 2 shows as an example the process of a complete reconfiguration from homogeneous flooding to homogeneous simple routing. The figure depicts a broker overlay consisting of brokers $B_1 \dots B_5$ and their connecting edges. To illustrate the process of reconfiguration, we added a label to every edge. The labels $S1$ to $S3$ express the causal relation between the edge reconfigurations. $S2$ directly depends on $S1$, and $S3$ directly depends on $S2$. The first possible reconfigurations are, thus, $r_{3,1} = [FL, SR]$, $r_{3,2} = [FL, SR]$ and $r_{4,5} = [FL, SR]$, because B_1, B_2, and B_5 are leaf brokers with an empty set $\mathcal{S}_{ij}$. All other edges have to apply flooding because the criterion of Eq. 1 is not fulfilled. If B_1, B_2 and B_5 have finished their reconfiguration (indicated by $S1$), the process is continued with $r_{4,3} = [FL, SR]$ and $r_{3,4} = [FL, SR]$ expressed by $S2$. This step can be performed because all edges in $S_{3,4}$ and $S_{4,3}$ are already using simple routing. Step $S3$ completes the reconfiguration. Now all edges are using simple routing.

Switching from Simple Routing to Flooding. Switching from simple routing to flooding $r_{ij} = [SR, FL]$ is controlled by B_i, which first starts to flood notifications via e_{ij} instead of matching them. Additionally, B_i drops its routing table and stops the processing of subscriptions that B_j is still sending. In the second step, B_i sends a message to B_j that it should stop sending subscriptions. With the processing of this message at B_j, the reconfiguration from simple routing to flooding is completed.

To avoid that $r_{ij} = [SR, FL]$ leads to a violation of Eq. 1 by any predecessor edges $\mathcal{P}_{ij}$, it is necessary that all of these edges use flooding. Since $\mathcal{P}_{ij}$ is empty for leaf brokers, they can always switch outgoing edges to flooding. For the example presented in Fig. 2, the switching from homogeneous simple routing to homogeneous flooding would be executed just in the reverse order.

5 Self-Optimizing Routing

Based on the results of Sect. 4, we are now able to use the reconfiguration mechanism to step-wise reduce the costs of routing. We first introduce a cost model in Sect. 5.1 and then describe how each broker can detect whether a reconfiguration of an edge is profitable or not in Sect. 5.2. In Sect. 5.3, we present the algorithm, which implements the self-optimizing routing.

5.1 Metrics

For optimization purposes we define a cost measure that directly depends on the *notification rate* ω_{ij}^N and the *subscription rate* ω_{ij}^S in the system at one moment in time[1]. An obvious approach for a cost measure C_{ij} for edge e_{ij} is:

$$C_{ij} = c_{ij} \cdot \omega_{ij}^N + c_{ji} \cdot \omega_{ij}^S \tag{2}$$

The cost coefficients c_{ij} and c_{ji} represent the costs for forwarding and processing one message. Depending on the optimization objectives, like optimizing network performance or load balancing, c_{ij} can be derived from metrics like CPU time, memory consumption, or forwarding latency.

To derive the global cost measure, we simply sum the costs of all edges:

$$C = \sum_{\forall e_{ij} \in E} C_{ij} \tag{3}$$

The self-optimizing routing tries to reduce this cost measure by adapting the routing algorithm. To achieve this, beside the current costs C_{ij} on one edge also the expected costs $\widehat{C}_{ij}$ after a reconfiguration of the routing algorithm are important. Similar to Eq. 2 one can define them by:

$$\widehat{C}_{ij} = c_{ij} \cdot \widehat{\omega}_{ij}^N + c_{ji} \cdot \widehat{\omega}_{ij}^S \tag{4}$$

How these expected message rates of notifications $\widehat{\omega}_{ij}^N$ and subscriptions $\widehat{\omega}_{ij}^S$ can be determined is discussed in the following section.

[1] Subscriptions are forwarded from B_j to B_i

5.2 Local Decisions

Now, we derive under which conditions the edge reconfigurations $r_{ij} = [FL, SR]$ respectively $r_{ij} = [SR, FL]$ reduce the costs of edge e_{ij} according to the cost model introduced before. To decide whether a reconfiguration is reasonable or not, the expected costs after a reconfiguration $\widehat{C}_{ij}$ must be compared to the current costs C_{ij}. It is obvious that if $\widehat{C}_{ij} < C_{ij}$ holds the cost of the link can be reduced by a reconfiguration. This implies the reduction of the global costs of the system because there is no influence on other edges in the system (cf. Sect. 5.3). Since the current costs C_{ij} can easily be determined by both brokers B_i and B_j, it is necessary to analyze which broker can derive the expected costs $\widehat{C}_{ij}$ for both directions of reconfiguration.

Flooding to Simple Routing. While using flooding on edge e_{ij}, broker B_i sends all notifications to B_j without considering B_j's interests. This can lead to a high amount of notifications that B_j has to discard because neither B_j itself or one of its neighbors connected by $\mathcal{S}_{ij}$ are interested in those notifications. If the number of discarded messages exceeds a certain level, it is reasonable to reconfigure the edge configuration to simple routing, i.e., to send subscriptions to B_i and therefore to reduce the number of notifications.

In order to compute the expected costs $\widehat{C}_{ij}$ that will occur in simple routing, the message rates $\widehat{\omega}^N_{ij}$ and $\widehat{\omega}^S_{ij}$ must be determined. Both values can be measured by broker B_j. On one hand, the expected subscription message rate $\widehat{\omega}^S_{ij}$ equals the sum of the local subscription rate and the aggregated message rates from all successor edges $\mathcal{S}_{ij}$. On the other hand, the expected notification rate $\widehat{\omega}^N_{ij}$ can easily be derived since the broker is already matching notifications before forwarding them to neighbors. Thus, the expected rate is equal to the interest rate.

Simple Routing to Flooding. If simple routing is applied on e_{ij}, broker B_j sends subscriptions to B_i that builds up a routing table to filter the notifications forwarded to B_j. If in some situation the effort for using simple routing is higher than for flooding, a reconfiguration is suitable. This requires that B_j is interested in almost all notifications forwarded by B_i or that the amount of subscription messages is significantly higher than the one of notifications.

To compute the expected costs $\widehat{C}_{ij}$ of using flooding, it is only necessary to determine the notification message rate $\widehat{\omega}^N_{ij}$ since $\widehat{\omega}^S_{ij} = 0$. The information about the expected flooding rate is only available on broker B_i because it is receiving all notifications from $\mathcal{P}_{ij}$ and of its local clients. This accumulated rate will appear on the overlay link e_{ij} if a reconfiguration is triggered.

Both reconfiguration criteria can be evaluated by the broker that was the coordinator of a routing reconfiguration before (cf. Sect. 4.2). Thus, it can be seamless integrated into an self-optimization algorithm, which is introduced next.

5.3 The Algorithm

If we combine all concepts introduced before, we can implement a self-optimizing algorithm consisting of three active steps that are periodically repeated:

1. Each broker checks if there are candidate edges that can potentially be reconfigured. Depending on the current routing algorithm, for such an edge e_{ij} either all edges in $\mathcal{S}_{ij}$ must use simple routing or all edges in $\mathcal{P}_{ij}$ must use flooding.
2. For each candidate edge, the broker checks if the reconfiguration is reasonable. This is true if the expected costs are lower than the current costs. To avoid oscillation in the optimization process, a reconfiguration is only triggered if the expected gain is higher than a predefined value (e.g. 5%).
3. For each reasonable reconfiguration, the reconfiguration process is triggered as described in Sect 4.2.

Beside the described active role, every broker implements listeners for the additional messages, which have been introduced for the reconfiguration process (Sect. 4.2). These listeners directly implement the two actions "stop sending subscriptions" and "start filtering". Furthermore, incoming and outgoing message traffic is monitored in order to update the message rates for cost calculations. To reduce noise, these rates are calculated by averaging over a bounded time window.

Major Properties. The described algorithm has the following major properties: (i) It is completely distributed, (ii) it requires only local knowledge, (iii) it does not depend on a complex interaction scheme, and (iv) it introduces only little additional traffic. While the first property is trivial, the second one can directly be derived from the discussion of local optimizations in Sect. 5.2. Properties (iii) and (iv) result from the described reconfiguration and optimization process in Sect. 4.2. There, we showed that any reconfiguration is processed by first evaluating the reconfiguration condition, second by changing the local behavior, and third by sending a signal to neighboring brokers. Besides these signaling messages, the only additional messages that have to be sent are the messages required to build up the routing table when switching an edge from flooding to simple routing. If the time between consecutive reconfigurations from flooding to simple routing is not shorter than the average subscription lifetime, no additional messages compared to simple routing are needed. Furthermore, there is no need of additional coordination since reconfigurations are done edge by edge and there is no possible configuration, where neighbor brokers will reconfigure the same edge at the same time: If flooding is active on an edge e_{ij}, only broker B_j is able to trigger a reconfiguration. If simple routing is used, only B_i is able to initialize a reconfiguration. Therefore, the self-optimizing routing algorithm assures a conflict-free adaptation of the system.

Locality. Interestingly, the reconfiguration of an edge does not influence the message traffic on other edges. Based on Sect. 4.2, a reconfiguration affects only

the rates of notifications and subscriptions on an individual edge. Therefore, cost reduction on a particular edge results in reduced global costs, too. This can be shown by examining a reconfiguration $r_{ij} = [FL, SR]$. We know that all edges in $\mathcal{S}_{ij}$ use simple routing and all edges in $\mathcal{P}_{ij}$ use flooding, otherwise the correctness criterion defined by Eq. 1 would be violated. In between, e_{ij} switches to simple routing and subscriptions are sent to B_i, but they are not forwarded to the edges in $\mathcal{P}_{ij}$ because they use flooding. On the other side, B_i filters incoming notifications before forwarding them to B_j, but there is no notification that appears additionally on edges in $\mathcal{S}_{ij}$ because B_i filters them already. These two facts show that $r_{ij} = [FL, SR]$ does not influence other edge costs. Similarly, we can argue for $r_{ij} = [SR, FL]$.

Optimality. In a static situation, all rates stay constant and local optimizations are repeated until no further reconfigurations are profitable. The optimization process leads to costs that are equal to or less than that of homogeneous routing. Because of the locality mentioned before it is obvious that the global cost measure can be reduced in each step of self-optimization. Thus, local optimality is guaranteed. Furthermore, it can be shown that also global optimality can be reached if certain constraints for the cost coefficients c_{ij} hold. Due to limited space, we cannot present the proofs here.

6 Evaluation

Now, we present the results of a discrete event-simulation of the proposed algorithm. The setup is based on a complete binary overlay tree with 127 brokers. There are $1,000$ different types of notifications that clients can publish or exclusively subscribe to. The type and the originating broker of publications or subscriptions are randomly chosen. The expected overall publication rate is set to $500s^{-1}$. The initial subscription birth rate is set to $10s^{-1}$ with an expected subscription lifetime of one minute. All three processes are stochastic and exhibit exponential distributions. All cost coefficients c_{ij} were set to 1 and the active role of the self-optimizing routing (Sect. 5.3) is called every 10 seconds. A reconfiguration is only triggered if the expected gain is higher than 5%. The results have been averaged over $1,000$ runs.

Figure 3 compares the global message traffic when using self-optimizing routing, homogeneous simple routing and homogeneous flooding. In order to show the adaptiveness of self-optimizing routing, we modify the behavior of subscribers as follows: After 400 seconds, the subscription birth rate is continuously increased up to $20,000s^{-1}$. This rate is reached approximately 850 seconds after simulation start. After a short period with a stable rate, it is then continuously reduced to $2,000s^{-1}$. This value is reached at about 1800 seconds.

During the phase of a low subscription birth rate, self-optimizing routing and simple routing show the same behavior because both use simple routing on all edges. In contrast to the behavior of simple routing, flooding indicates a constant message traffic. Due to the property of flooding, all notifications are routed in all

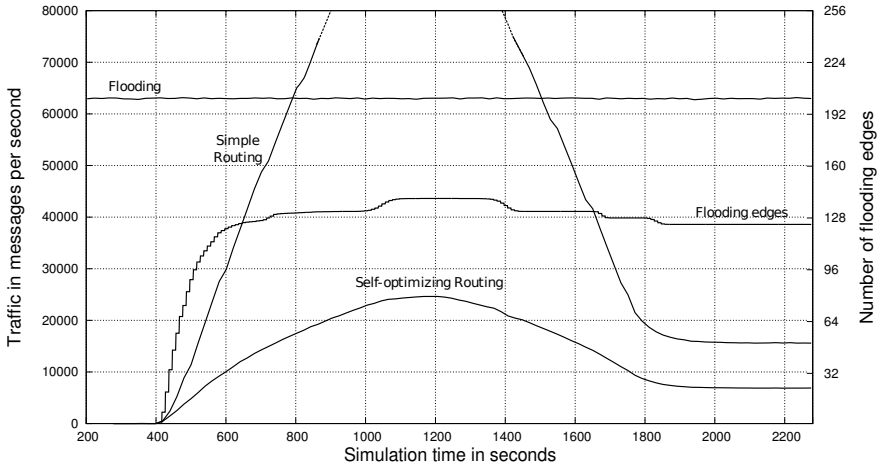

Fig. 3. Self-optimizing routing compared to simple routing and flooding

possible directions. Therefore, flooding is independent of the subscription birth rate. Although flooding shows a bad performance in order to deliver all messages in the system, simple routing gets even worse in case that the subscription birth rate exceeds a certain threshold. Approximately, after 800 seconds the curve indicating simple routing intersects the one for flooding. As a matter of fact, the curve representing the message traffic in self-optimizing routing resides always under the curve for simple routing and flooding. Due to its adaptive behavior, the algorithm reacts to the increase of the subscription birth rate by switching edge-wise beginning with the leaf brokers from simple routing to flooding. One may notice that the curve for reconfigurations increases strongly between 400 and 600 seconds. In this period, the algorithm detects high subscription rates on several edges and switches them to flooding. In total, the algorithm switched 140 out of 252 edges to flooding in order to reduce the message traffic. All remaining edges still use simple routing. A reverse reconfiguration phase can be seen after 1200 seconds when the subscription birth rate is reduced. Now some of the edges switch back to simple routing to avoid unnecessary delivery of notifications.

7 Conclusions

In this paper, we presented an optimization algorithm, which continually adapts the routing configuration of a content-based publish/subscribe system to reduce its cost. The presented algorithm works decentralized with little traffic overhead and reduces the costs with respect to the introduced cost model. The optimization is based on the introduction of hybrid routing algorithms, which allow to combine several routing algorithms in one broker overlay network. Before introducing this approach, the same routing algorithm had to be applied by all

brokers and it had to be selected before system start. Our approach is especially advantageous in dynamic scenarios, where the clients' behavior varies over time. However, it is also useful in static scenarios, where the information necessary to determine the optimal routing algorithm is not available before runtime.

Although we used hybrid routing algorithms composed of *simple routing* and *flooding* as an example, our approach works with any combination of at least two routing schemes. Therefore, we want to extend the optimization algorithm to include other routing algorithms such as identity-based, covering-based, and merging-based routing. Additionally, it seems interesting to include advertisements, which are issued by producers to indicate the notifications they will potentially publish in the future. In this case, optimization will be applied on two layers corresponding to notification and subscription routing. Despite the extensions in the field of routing, our future work also addresses algorithmic challenges to avoid unnecessary reconfigurations even in highly dynamic situations. Therefore, the prediction of expected message rates and the consideration of reconfiguration costs will be investigated.

References

1. S. Bittner and A. Hinze. Dimension-based subscription pruning for publish/subscribe systems. In A. Hinze and J. Pereira, editors, *Proceedings of the 5th International Workshop on Distributed Event-Based Systems (DEBS'06)*, pages 25–25, Lisbon, Portugal, July 2006. IEEE, IEEE.
2. S. Bittner and A. Hinze. Pruning Subscriptions in Distributed Publish/Subscribe Systems. In *Proceedings of the Twenty-Ninth Australasian Computer Science Conference (ACSC 2006)*, pages 197–206, Hobart, Australia, Jan. 2006. ACS.
3. S. Bittner and A. Hinze. Subscription Tree Pruning: A Structure-Independent Routing Optimization for General-Purpose Publish/Subscribe Systems. Technical Report 01/2006, Computer Science Department, University of Waikato, Jan. 2006.
4. A. Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks.* PhD thesis, Politecnico di Milano, Milano, Italy, Dec. 1998.
5. P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.
6. L. Fiege, G. Mühl, and A. Buchmann. An architectural framework for electronic commerce applications. In *Proceedings of the 2001 Annual Conference of the German Computer Society (Informatik 2001)*, pages 928–938, Viena, Austria, September 2001.
7. M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann. Self-organizing broker topologies for publish/subscribe systems. In L. M. Liebrock, editor, *Proceedings of the 22nd Annual ACM Symposium on Applied Computing (SAC'07)*, pages 543–550. ACM, Mar. 2007.
8. G. Mühl, L. Fiege, and P. R. Pietzuch. *Distributed Event-Based Systems.* Springer-Verlag, Aug. 2006.
9. P. R. Pietzuch. *Hermes: A Scalable Event-Based Middleware.* PhD thesis, Computer Laboratory, Queens' College, University of Cambridge, Feb. 2004.
10. A. Schröter, G. Mühl, J. Richling, and H. Parzyjegla. Adaptive routing in publish/-subscribe systems using hybrid routing algorithms. In *ARM '08: Proceedings of the 7th workshop on Reflective and adaptive middleware*, pages 51–52, New York, NY, USA, 2008. ACM.