On Quality-of-Service and Publish-Subscribe

Stefan Behnel
Databases and Distributed Systems Group,
Technische Universität Darmstadt (TUD),
Darmstadt, Germany,
behnel@dvs1.informatik.tu-darmstadt.de

Ludger Fiege
Siemens AG,
Munich, Germany,
ludger.fiege@siemens.com

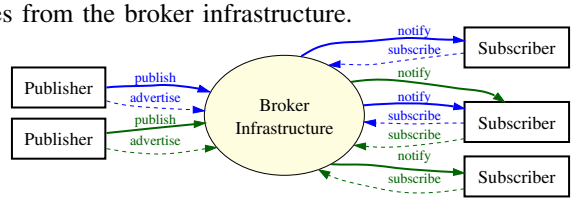
Gero Mühl
Communication and Operating Systems,
Technische Universität Berlin,
Berlin, Germany,
gmuehl@cs.tu-berlin.de

***Abstract*— Publish-subscribe is a powerful paradigm for distributed communication based on decoupled producers and consumers of information. Its event-driven nature makes it very appealing for large-scale data dissemination infrastructures. Various architectures were proposed in recent years that provide very diverse features. However, there are few well-defined metrics in the publish-subscribe area that would allow their evaluation and comparison.**

In this paper, we provide a broad overview of relevant quality-of-service metrics and describe their specific meaning in the context of distributed and decentralized publish-subscribe systems. Our goal is to provide a common base for future evaluations of emerging systems and for the design of quality-of-service aware publish-subscribe infrastructures.

I. Introduction

The system model of the publish-subscribe communication paradigm is surprisingly simple. The figure shows its three roles: publishers, subscribers and brokers. Publishers (aka producers) provide information, advertise it and publish notifications about it. Subscribers (aka consumers) specify their interest and receive relevant information when it appears. Brokers mediate between the two by selecting the right subscribers for each published notification. Additionally, we use the term "client" for both publishers and subscribers to distinguish their roles from the broker infrastructure.



The two major approaches for notification filtering are *subject-based* publish-subscribe, where consumers subscribe to a subject that producers explicitly assign to their notifications, and *content-based* publish-subscribe, where subscriptions include more general filters on the actual content of notifications.

An important property of the publish-subscribe model is the level of abstraction at which publishers and subscribers communicate. They are not aware of organization and size of the system. There can be a single centralized broker, a cluster of them or a distributed network of brokers. All that clients see is their specific brokers through which communication partners are self-selecting by interest. This makes this model appealing for highly scalable systems that must hide varying complexity and adaptable infrastructures from the participants.

Designing efficient, scalable infrastructures has been a major research interest in recent years. Various systems were proposed, including [4, 18, 22]. However, they have rather diverse characteristics and none of them answers all requirements of the various possible applications of publish-subscribe. This paper tries to provide a common ground for the comparison and evaluation of these systems. To this end, we survey the relevant quality-of-service metrics and describe their meaning within the very specific context of the publish-subscribe model. To the best of our knowledge, this is the first comprehensive evaluation of QoS metrics in the publish-subscribe area.

The publish-subscribe model suggests a number of different quality-of-service metrics. Some are related to subscriptions and single notifications while others describe end-to-end properties of flows of notifications. The following sections describe the different metrics and their meaning when requested by publishers or subscribers. Each description starts with a definition (☞) of the metric with respect to subscriptions, notifications and advertisements. Their implementation in a distributed publish-subscribe system has mainly two dimensions: The topological organization of the brokers and the local decisions of each broker regarding filtering, scheduling, etc. They are briefly summarized in figure 1.

II. QoS at the global infrastructure level

End-to-end latency, bandwidth and delivery guarantees form low-level properties of the broker infrastructure. In a centralized infrastructure in which the client connections also implement QoS, there are ways to impose hard limits on them. In any less predictable environment, especially distributed multi-hop infrastructures, they should not be understood as real-time guarantees. Here they become probabilistic options or even hints about preferences of clients. Note that even hints can be helpful to the infrastructure if it has to determine which notifications to drop from overfull queues or which broken inter-broker connection to repair first.

A. Latency

☞ *Subscriptions* - Subscribers request a publisher that is within a maximum latency bound.

The end-to-end latency between producers and consumers depends on the number of broker hops between them, the travel time from hop to hop and the time it takes each broker to forward a notification.

Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'06)
0-7695-2541-5 /06 $20.00 © 2006 IEEE

QoS Metric	*Topology Impact*	*Local Broker Decisions*
Latency	end-to-end network latency and hop count, reorganization delays, global load distribution, adaptation to physical topology, degree of freedom in routing	time complexity of filter evaluation, local routing choices
Bandwidth	end-to-end bandwidth, global load distribution, path independence, adaptation to physical topology, degree of freedom in routing	local routing choices
Message priorities	topology shortcuts	local scheduling
Delivery guarantees	reliability, reorganization, path redundancy, trusted or reliable subgraphs	persistency and caching, routing decisions based on reliability, trust or reputation
Selectivity of subscriptions	adaptation to subscription language (e.g. subject grouping)	decidability, local load
Periodic/sporadic	-	conversion
Notification order	single/multi-path delivery, consistency during reorganization	reordering
Validity interval	deterministic delivery paths for follow-ups	message drops after delay or on follow-ups
Source redundancy	path convergence	identity decision
Confidentiality	trusted subgraphs	encrypted filtering
Authentication and integrity	-	client authentication, access control

Fig. 1. Relation of topologies and local broker decisions to the QoS metrics: Impacts and implementation choices

In a centralized system, the minimum travel time between broker and clients provides a hard lower bound and the additional forwarding time depends on the broker load. Even in a distributed broker infrastructure, measured lower bounds can give hints if a requested QoS level is achievable at all. In general, however, they do not allow for absolute guarantees. One way of dealing with latency requirements is by pre-allocating fixed paths between senders and receivers. This avoids the overhead of having to establish connections on request and allows for meaningful predictions.

A further speed-up in content-based systems can be achieved by tagging notifications and merging them into channels [11]. This simplifies the routing on each broker on the path by avoiding content-based filtering. Note that this approach effectively maps content-based filtering to subject-based filtering, which is not always possible as it can lead to state explosion [14]. The most efficient form obviously is a mapping to native IP-Multicast [8].

Systems could further reduce the path length by skipping those brokers that only forward a stream to one or very few connections. This obviously requires a certain freedom in the routing choices of brokers. Still, the observed latency may vary considerably over time, as Internet paths commonly serve multiple concurrent streams. The optimal case is a physical network that supports native quality-of-service allocation and IP-Multicast.

B. Bandwidth

- ☞ *Advertisements* - Producers specify upper/lower bounds for the stream they produce.
- ☞ *Subscriptions* - Subscribers restrict the maximum stream of notifications they want to receive.

The overall bandwidth used by the system depends on the throughput per broker and the size of each notification. Today's Internet-level connections tend to be sufficiently dimensioned to allow many concurrent high-traffic streams, but they usually do not provide physical quality-of-service and bandwidth provisioning. This holds especially when streams cross the borders of autonomous systems.

Therefore, bandwidth requirements should rather be regarded at a per-broker level. If each broker knows the bandwidth that it can make locally available to the infrastructure, this gives an upper bound for the throughput of a path. Although not necessarily accurate, such an upper bound allows to route notifications based on the highest free bandwidth on the neighbouring brokers. It can be used to avoid high-traffic paths and to do local traffic optimization. If channel merging is applied (as for latency), the channels can be tested for their end-to-end capacity. However, the observed bandwith in general purpose networks may show high variations over time that cannot be foreseen.

The subscriber's point of view is most important in mobile settings. Here, it makes sense to let the infrastructure restrict the delivery bandwidth to reduce the resource consumption of subscribers [15].

C. Message priorities

- ☞ *Notifications* - Producers specify relative priorities between their own notifications or their absolute priority compared to other (foreign) notifications.
- ☞ *Subscriptions* - Subscribers specify the relative priority between their subscriptions.

As with latency and bandwidth, message priorities have both a per-broker side and an end-to-end side to them. They are, however, easier to establish in a distributed broker network than latency bounds, as their conception does not aim to provide absolute or real-time guarantees.

Priorities between notifications can be used to control the local queues of each broker which will eventually lead to their end-to-end application along a path. Again, channels can be used to shorten the path for high-priority notifications, the extreme case being to send them directly from the system entry point to the recipients. More commonly, however, notifications will be allowed to overtake those with a lower priority during the forwarding and filtering process at each broker, subject to a weighted scheduling policy. This underlines the importance of their per-broker part.

D. Delivery guarantees

- ☞ *Subscriptions* - Subscribers specify which notifications they must receive, which are less important to them, and where duplicates matter.
- ☞ *Notifications, Advertisements* - Producers specify if subscribers must receive certain notifications.

Implementations can be as simple as a tag for notifications that *may be dropped* on the delivery path. This is even trivially merged from multiple subscriptions along the delivery path by a simple boolean "and". A combination with message priorities allows the least important message to be dropped first.

More demanding guarantees regard the completeness and duplication of delivery. Subscribers can receive notifications *at least once*, *at most once* or *exactly once*, the latter being the combination of the first two. If the infrastructure is not reliable itself, the first requirement can be achieved by meshing which increases the delivery probability at the cost of generating duplicates and thus increasing the message overhead. A request for at most one delivery encourages either single path delivery or duplicate filtering before the arrival at the subscribers.

In infrastructures with unreliable brokers, reputation systems or availability histories may allow to route notifications through more reliable brokers. This lowers the chance of brokers failing during delivery or maliciously suppressing notifications. However, this is rather a hint than a guarantee since a history does not allow a reliable prediction of the future availability or behaviour of network and brokers.

Another problem with delivery guarantees regards temporarily disconnected subscribers [6], e.g. in a mobile environment. The broker infrastructure must store notifications that were guaranteed to be delivered at least once until the subscribers become available again. Note that this may be after an arbitrarily long time or never, so in practice, the infrastructure will only store notifications for a sensible time interval.

Finally, it is possible to define a *quorum*, i.e. to specify the minimum or maximum number of subscribers that must receive a notification or the minimum or exact number of publishers that a subscriber wants to subscribe to. In a decoupled environment like publish-subscribe systems, where publishers and subscribers are not supposed to know anything about each other, this measure should only be available at an administrative level, e.g. within scopes [12].

III. QoS AT THE NOTIFICATION AND SUBSCRIPTION LEVEL

A number of QoS properties touch the semantics of subscriptions and notifications, including periodic or sporadic delivery, priority and order of delivery, limited validity and redundancy of producers. Security issues like authentication or confidentiality also fall into this scheme. A very important factor is the selectivity of subscriptions.

A. Selectivity of Subscriptions

- ☞ *Subscriptions* - Subscribers define their subscriptions in a specific language.

Subscriptions can be expressed in different classes of languages. Simple subscriptions can contain a subject for identity matches, whereas more complex ones can support attribute matches, ranges or even Turing complete computer programs.

All of these languages have a specific level of complexity with respect to filter identity tests and merging, distributed matching, global message overhead and false positives on delivery. The filter language therefore represents a tradeoff between local overhead at brokers or subscribers and distributed overhead inside the broker infrastructure. In general, more complex languages can reduce the number of false positives by allowing more accurate subscriptions, but tend to make filter optimizations and distributed matching more difficult.

The language is not necessarily the same within the entire infrastructure. Scoped subgraphs [12] may decide to provide languages internally that differ from the outside capabilities and then convert between the two at the interfaces. Similarly, a broker may decide to accept complex subscriptions from its local subscribers and forward simplified versions that are understood by remote brokers or more suitable for the topology. If it then filters incoming notifications based on the more expressive subscriptions, it can increase the satisfaction level of the local subscribers.

B. Periodic or sporadic delivery

- ☞ *Advertisements* - Producers advertise their output.
- ☞ *Subscriptions* - Subscribers specify which notifications they want to receive sporadically and which they need periodically.

This captures the difference between an interest in changes of information and the information itself. Periodically published notifications become a data flow that represents the status of requested information at the moment of each publication. Sporadically published notifications occur only when this information changes. One way of looking at them is as prefiltered events that pass when the data change exceeds a certain threshold. This is an interesting option for reducing the amount of data sent by accepting a certain inaccuracy [15].

The infrastructure may either match subscriptions to corresponding publishers or try to emulate the requested state by itself. Periodic notifications may be emulated by storing and repeating the latest sporadic notification. Sporadic delivery can be based on a configurable threshold that blocks the delivery of periodically published static values. Note that in a content-based system with arbitrary filters, the required comparison of notifications may be restricted to an identity test or may not be possible at all.

C. Notification order

- ☞ *Advertisements* - Producers advertise for which of their notifications the ordering matters.
- ☞ *Subscriptions* - Subscribers specify which matching notifications they want to receive in order.

The order in which notifications arrive may or may not be relevant. In many cases, ordering is easy to achieve by either using centralized ordering, ordered transports (ATM, TCP) or by letting the producer (or its broker) impose an explicit order and sorting the notifications on delivery.

Special care must be taken if the broker topology is allowed to change during the delivery process, as this may impact the order in which notifications arrive and may even result in message loss. Since ordered delivery does not imply guaranteed delivery, a very efficient solution is to deliberately drop notifications if a successor has already been delivered. If this is not acceptable, notifications must be reordered before delivery, which may introduce arbitrarily long delays.

The distributed ordering of events coming from different sources is another problem. For content-based subscriptions, it is even hard to define a meaningful ordering in this case. It is therefore largely dependent on the subscription language and the application if such an order is applicable. A generic approach is the deployment of a dedicated, central broker to enforce a global ordering, which can in turn limit the scalability of the overall infrastructure.

D. Validity interval

- ☞ *Notifications, Advertisements* - Producers advertise or specify a timeout for their notifications, or a successor message that renders them irrelevant.

It is important for the infrastructure to know how long a notification stays valid, either specified in terms of time or infered by the arrival of later messages. A validity based on a hop count would be meaningless in the decoupled publish-subscribe model. If only the most recent event is of interest, the validity specification by follow-up messages is a particularly efficient approach. It allows the infrastructure to reorder and shorten its queues in high traffic situations.

E. Source redundancy

- ☞ *Subscriptions* - Subscribers request redundant sources for the same event.

Redundant sources increase the fault tolerance and allow the subscribers to double check events. In some cases, they may be sufficient to replace guaranteed delivery. Note that a request for redundancy requires the notifications or advertisements of publishers to be comparable. Complex subscription languages may hinder or prevent this (decision problem[1]).

F. Confidentiality

- ☞ *Subscriptions* - Subscribers encrypt their subscriptions or send them only to trusted brokers.
- ☞ *Notifications* - Publishers connect only to trusted brokers or send (partially) encrypted notifications.

Confidentiality is obviously achievable in (sub-)networks of only trusted brokers [13]. Similarly, messages can be hidden from untrusted brokers on the delivery path by using standard encryption mechanisms between trusted brokers. Untrusted brokers are then forced to broadcast to all their neighbours. Apart from higher load at those brokers, this also introduces potentially large numbers of duplicates in the system.

Confidentiality becomes difficult where untrusted brokers must participate in the matching process. This is a minor problem in subject-based publish-subscribe, where the low selectivity of subscriptions makes the actual content of notifications opaque to the matching process. It can just as well be encrypted, which may be exploited within untrusted broker networks. The information gain of untrusted brokers then depends on the amount of information revealed by the subject. The selectivity of subjects can therefore be seen as a tradeoff between message overhead and reveiled information. However, malicious brokers can still suppress messages in this case.

Content-based matching requires much higher insight into the content of notifications. A number of recent publications from the database area show ways for matching encrypted data against certain queries [1, 2] without revealing any of the two. Publish-subscribe systems could apply similar schemes to allow blind matching on untrusted brokers. However, solving this problem for arbitrary queries and data without reveiling any information about them is likely impossible.

G. Authentication and Integrity

- ☞ *Subscriptions* - authenticated by Subscribers.
- ☞ *Notifications* - authenticated by Publishers.

If publishers want to enforce access control mechanisms [3], it becomes necessary for subscribers to authenticate their subscriptions using techniques like digital signatures. On the other side, publishers can sign or watermark their notifications to assure data integrity. The evaluation can then either be done end-to-end or within the broker infrastructure.

IV. Evaluation of Publish-Subscribe Systems

This section briefly evaluates a number of existing systems by their quality-of-service characteristics. Following an important development in recent years, we focus on overlay networks as distributed broker infrastructure [17, 23]. They provide interesting new filtering approaches as well as a design simplification for distributed publish-subscribe systems. First attempts were targeted at replacing application-level multicast, while later approaches support content-based filtering.

A. Multicast and subject-based approaches

The *selectivity* of subjects (or multicast groups) is relatively low. This means that subscribers are left to do their own filtering after delivery (mailing lists are a good example). On the other hand, if expensive subscription languages render matching and optimizations difficult, it may be worth considering a combination with subjects to provide a fast fallback through subject matches. Such an optimization can considerably reduce the overhead of messages and filtering.

Overlay multicast allows for two general approaches for publish-subscribe systems. They can build group dissemination trees inside a global overlay (overlay internal), or they can build a separate broadcast overlay for each group (overlay external). The major advantage of external multicast is that each group overlay only has to scale with the size of a group. A disadvantage is the increased overhead of maintaining multiple independent overlays simultaneously [5].

[1] http://en.wikipedia.org/wiki/Entscheidungsproblem

Overlay internal multicast uses features of the respective overlay to emulate or establish groups within the overlay itself. Especially the class of key-based routing overlays (KBR, commonly used in distributed hash tables) provides efficient intrinsics for lookup operations. The obvious approach is therefore to have notifications and subscriptions meet at the node that a lookup yields for the multicast address (or subject), the so-called rendez-vous node. Examples are Hermes [17] and Scribe [20], which deploy the Pastry KBR overlay [19].

Nodes in the internal scheme must forward messages they are not interested in. Intermediate schemes, as proposed for Scribe [20], can try to skip nodes by merging their children. Since this deviates from the original overlay topology, additional maintenance is necessary to handle these optimizations. Consequently, this approach can also be seen as the construction of a new overlay where the bootstrap overhead is reduced by reusing known nodes. We can therefore suspect the transition between the two methods to be rather seamless.

Latency: The rendez-vous node in internal multicast represents a bottleneck, although it may not even be interested in the subject. It must see all notifications and subscriptions for the group to assure correctness and completeness of delivery. In general, the total number of subscriptions of a node's neighbours is likely to be higher near the rendez-vous nodes than elsewhere in the network. This increases the load of these important nodes and therefore the overall latency.

External group overlays, on the other hand, have the same size as the group of interest and commonly feature very good load balancing. This minimizes the end-to-end hop count, which most likely becomes smaller than in the global overlay. Obviously, this reduces the number of required routing decisions and minimizes the end-to-end latency for the given topology. Note that the group overlays are independent of the global overlay and may even use a different topology to optimize for their group size and specific requirements.

Delivery guarantees: Delivery guarantees can exploit the fact that all participants in an external group overlay are interested in all messages. This reduces the cost of deliberate duplications which can be used to increase the probability of delivery, even under topological reorganization. The redundancy that is already available in a resilient overlay can directly be exploited to assure the broadcast delivery to all live parties as long as the network stays connected. If duplication is undesirable, most broadcast schemes for KBR overlays can efficiently avoid message duplication as long as there are no reorganizations during the delivery process [23].

If a subset of the group members caches the received notifications, a node that missed notifications during a temporary failure can try to ask any of the other members for passed notifications, possibly using a random walk or an attenuated broadcast scheme. Similar ideas have been presented in [6].

The higher load near rendez-vous nodes may impact the delivery in internal multicast. Message duplication and loss is more likely than in the external scheme, as the probability of topological reorganization increases with the number of brokers. On the other hand, a higher number of brokers provides higher redundancy in the overall network. Algorithms can exploit this to re-enhance the probability of delivery.

Confidentiality: Only the external scheme effectively prevents non-subscribers from receiving notifications. If some form of access control is available for the subscriptions, nodes can be prevented from joining the network. This enables networks of trusted brokers. In internal multicast, untrusted brokers must either broadcast encrypted notifications or the infrastructure can reveil the subject and only encrypt the content of the notification.

B. Hermes and IndiQoS

Hermes [17] was the first to exceed the limitations of overlay multicast by implementing a form of content-based publish-subscribe (named *type and attribute based filtering*) on top of overlay networks. It deploys the Scribe internal multicast scheme, but additionally installs content-based filters along the delivery path that filter on additional attributes of the notifications. Filter merging is possible along the path to reduce the status overhead per node.

IndiQoS [4] augments the Hermes design with QoS awareness. QoS requirements are expressed as additional attributes of subscriptions and advertisements that are evaluated during forwarding. As further improvement, IndiQoS replicates rendez-vous nodes and selects one based on QoS subscriptions. This is obviously payed with a multiplication of the message overhead per notification.

Selectivity: Filtering on arbitrary attributes along the multicast paths substantially increases the selectivity of subscriptions and therefore reduces the number of false positives. This is bought with an increased message overhead for filter updates when compared to multicast systems.

Confidentiality: Hermes brokers can exploit the content of notifications in different granularities. The system can either reveil subjects and certain attributes to untrusted brokers or encrypt everything and require them to broadcast.

Latency: Only IndiQoS allows subscribers to specify maximum latencies. Brokers then forward subscriptions according to latencies known from prior advertisements. The rendez-vous node replication then yields a lower average latency.

C. Choreca

Choreca [23] is an extension of the REBECA system [14, 18], a content-based publish-subscribe system that exemplifies filter merging and distributed administration. Choreca routes REBECA's content-based subscriptions and notifications over Chord broadcast trees [9], one rooted in each publisher. The binomial redundancy of these trees avoids any single point of failure and allows to exploit filter similarities to reduce the forwarding overhead.

Selectivity: REBECA and Choreca support arbitrary filters as subscriptions, which allows for a high selectivity and therefore low rate of false positives. On the downside, highly expressive filters may prevent filter merging.

Delivery guarantees: The path redundancy in Choreca's Chord graph lies within $O(\log N)$. To assure the eventual delivery of a notification in the case of a lower number of link failures, it is sufficient to reroute it through a different neighbour than the failed next hop. This allows Choreca to provide a very high probability for delivery.

V. Related work

We presented a general overview of relevant quality-of-service metrics in the publish-subscribe area. There have been few publications on this topic so far. The IndiQoS system [4] distinguishes a *content profile* (such as the precision of a sensor) and a *QoS profile*, allowing to request periodic or sporadic delivery, as well as bandwidth requirements and latency bounds. No other metrics are considered.

In a section of a well-known paper by Eugster etal. [10], the authors refer to persistence, transactions and priorities as relevant metrics. In this paper, we grouped persistence under the more general concept of delivery guarantees. While message priorities have found their place, transactions are not regarded. They place very high requirements on the broker infrastructure and it is generally questionable if they are suitable for a decoupled system model like publish-subscribe.

Outside the field of publish-subscribe, there is a large body of literature on quality-of-service in the networking and Internet environment [24], especially regarding differentiated services (DiffServ) and resource reservation (RSVP etc.). Another area of interest is messaging middleware, including CORBA [16] and JMS [21], that support quality-of-service levels for their communication services. The selection of metrics presented in this paper largely follows previous work in these areas. The application to the publish-subscribe model, its roles and distributed infrastructures, however, has not been considered before in a comparatively extensive way.

VI. Conclusion

In large publish-subscribe systems with thousands of participants we will necessarily face very diverse demands from different users. Different applications have similarly diverse requirements. This calls for quality-of-service support in infrastructures and for means to express requirements and system capabilities in a meaningful and well-defined way.

Our analysis provides extensive insights into the diversity of requirements in publish-subscribe systems. We give a comprehensive overview of relevant quality-of-service metrics and describe their application to the publish-subscribe model, its roles and selected implementations. A general side effect of this model allows the requirements of subscribers to override the advertised requirements of publishers. The infrastructure has the freedom to ignore the latter if all subscribers agree to accept lower quality levels. Publisher requests therefore become more of a hint or default to the infrastructure.

This paper provides a solid foundation for future work on scalable publish-subscribe services. The described quality-of-service metrics allow their comparison and the definition and integration of forwarding domains that adhere to user-provided QoS parameters.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proc. of the ACM SIGMOD Int. Conference on Management of Data*, Paris, France, June 2004.

[2] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *WPES '03: Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, 2003.

[3] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody. Role-based access control for publish/subscribe middleware architectures. In DEBS03 [7].

[4] N. Carvalho, F. Arajo, and L. Rodrigues. Scalable QoS-based event routing in publish-subscribe systems. Technical report, Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa, Lisbon, Portugal, Feb. 2005.

[5] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast using peer-to-peer overlays. In *INFOCOM 2003*, 2003.

[6] M. Cilia, L. Fiege, C. Haul, A. Zeidler, and A. Buchmann. Looking into the past: Enhancing mobile publish/subscribe middleware. In DEBS03 [7].

[7] *Second Intl. Workshop on Distributed Event-based Systems (DEBS'03)*, San Diego, CA, USA, June 2003. ACM Press.

[8] S. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford, CA, USA, 1991.

[9] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured P2P networks. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, USA, Feb. 2003.

[10] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[11] L. Fiege. *Visibility in Event-Based Systems*. PhD thesis, Technical University of Darmstadt, Darmstadt, Germany, 2005.

[12] L. Fiege, M. Cilia, G. Mühl, and A. Buchmann. Publish/subscribe grows up: Support for management, visibility control & heterogeneity. *IEEE Internet Computing: Special Issue - Asynchronous Middleware and Services*, January 2006.

[13] L. Fiege, A. Zeidler, A. Buchmann, R. Kilian-Kehr, and G. Mühl. Security aspects in publish/subscribe systems. In A. Carzaniga and P. Fenkam, editors, *Third Intl. Workshop on Distributed Event-based Systems (DEBS'04)*, Edinburgh, Scotland, UK, May 2004. IEEE.

[14] G. Mühl, L. Fiege, and A. P. Buchmann. Filter similarities in content-based publish/subscribe systems. In H. Schmeck, T. Ungerer, and L. Wolf, editors, *International Conference on Architecture of Computing Systems (ARCS)*, pages 224–238, Karlsruhe, Germany, 2002.

[15] G. Mühl, A. Ulbrich, K. Herrmann, and T. Weis. Disseminating information to mobile clients using publish/subscribe. *IEEE Internet Computing*, 8(3), May 2004.

[16] Object Management Group. CORBA notification service, version 1.0.1. OMG Document formal/2002-08-04, 2002.

[17] P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *Proc. of the 1st Int. Workshop on Distributed Event-based Systems (DEBS'02)*, Vienna, Austria, July 2002. IEEE Press.

[18] Rebeca Event-Based Electronic Commerce Architecture. http://www.gkec.informatik.tu-darmstadt.de/rebeca/.

[19] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale Peer-to-Peer systems. In *Proc. of the Int. Middleware Conference (Middleware2001)*, Nov. 2001.

[20] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In J. Crowcroft and M. Hofmann, editors, *Proc. of the 3rd Int. Workshop on Networked Group Communications (NGC'01)*, London, UK, 2001.

[21] Sun Microsystems, Inc. Java Message Service (JMS) Specification 1.1, 2002.

[22] W. W. Terpstra, S. Behnel, L. Fiege, J. Kangasharju, and A. Buchmann. Bit Zipper Rendezvous - Optimal Data Placement for General P2P Queries. In *Proc. of the 1st Int. Workshop on Peer-to-peer Computing and Databases*, Heraklion, Crete, Mar. 2004.

[23] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. Buchmann. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In DEBS03 [7].

[24] W. Zhao, D. Olshefski, and H. Schulzrinne. Internet quality of service: An overview. Technical Report CUCS-003-00, Columbia University, Feb. 2000.