Revisiting Content-Based Publish/Subscribe

Costin Raiciu David S. Rosenblum Mark Handley
Department of Computer Science
University College London
{c.raiciu|d.rosenblum|m.handley}@cs.ucl.ac.uk

Abstract

Content-based publish/subscribe is a very appealing interaction model that has attracted intense efforts from the research community in the quest to obtain "Internet-wide" scalability. Despite many promising proposals, wide-area applications using content-based publish/subscribe are yet to be deployed. In this paper, we list possible reasons for this lack of adoption and propose a series of approaches to remedy this state of affairs. To simplify the problem tackled, we argue for the separation of content-based matching and event-routing. To cope with application diversity, we propose the development of configurable solutions for parts of the problem space and provide a proof-of-concept solution for content-based matching.

Keywords: Content-Based Publish/Subscribe, Layering, Configurability

1 Introduction

Content-based publish/subscribe (CBPS) is a very appealing interaction model that has attracted intense efforts from the research community in the quest to obtain "Internet-wide" scalability [4, 1, 15]. Despite many proposals, wide-area applications using CBPS are yet to be deployed.

In this paper, we attempt to understand the reasons for this lack of deployment and proposes a series of remedies. First, we propose decomposing the problem of content-based publish/subscribe into its constituent sub-problems, namely content-based matching and event delivery. These are then solved separately. Although controversial at first sight, this approach allows us to focus better on the two simpler facets of the problem without any important losses of performance or functionality. Indeed, content-based matching solutions can be combined with event delivery solutions to yield CBPS solutions.

The applications suitable for CBPS are extremely diverse. Moreover, static[1] solutions that could scalably accommodate all these applications are provably impossible to create. We further advocate that solutions should be designed to be easily configurable or should adapt to application characteristics. In support of this claim, we provide a proof-of-concept configurable architecture for content-based matching.

This paper is structured as follows. In Section 2, we discuss the reasons for the disappointing lack of adoption of content-based publish/subscribe. Following in Section 3, we argue for the separation of content-matching and event delivery. In Section 4, we advocate the development of configurable architectures and propose a simple design as a proof-of-concept. Section 5 discusses issues related to the deployment of CBPS solutions, concluding in Section 6 with a summary of arguments.

2 Current State of Affairs

Experience gained during a literature and application review we have conducted, reveals the following (perhaps obvious) reasons for the lack of wide-area applications that use CBPS nowadays: a) complexity of the general CBPS problem, b) heterogeneity of the applications suitable for CBPS, and c) lack of wide-area deployment of CBPS solutions. Let us expand the above.

2.1 Complexity of CBPS

The complexity of content-based publish/subscribe follows from the difficulty of solving its related sub-problems, content-based matching and event delivery. Content-based matching is the problem of finding all the subscriptions that match a given notification. Event delivery is the task of delivering the notification to the set of interested subscribers selected with content-based matching.

A first indicator of the difficulty of content-based matching is its tight relationship with the problem of supporting enhanced queries in large-scale networks, currently a very active research field (see the survey by Risson et al. [20]). Furthermore, a simple analysis of the content-based matching problem shows that in the worst case, a solution to it cannot be scalable on all dimensions. Let H be the maximum number of routing hops a notification visits until it is

[1] Static means that the architecture does not adapt to its input.

Table 1. Application characteristics

	Online Games [7, 9]	*RSS Feeds* [11]	*Stock Quotes* [22]	*Security Alerts* [6]	*Location Based Services* [5]
Expressiveness	range matches	keywords	range matches	set inclusion	spatial queries
Publisher Bandwidth	30kbps	small,irregular	$\sim$ 100Mbps	very small	1kbps
Maximum Latency	0.1s-1s	$\sim$10s	$\sim$0.01s	$0.1 - 1$s	$0.1 - 1$s
Subscription clustering	multimodal	power law	power law	power law	multimodal
#Subscribers	$\sim 10^2 - 10^3$	$\sim 10^6$	$\sim 10^3$	$\sim 10^6$	10^6
Subscribe frequency	high	low	medium	low	low
#Publishers	$10^2 - 10^3$	10^4	$1 - 10$	10^3	10^6
Publish frequency	$\sim 10^3 - 10^4$/s	10^2/s	10^3/s	irregular bursts	$10^{-1} - 1$/s

matched against all relevant subscriptions, R the replication rate of subscriptions and N the number of nodes in the system. Define I_r and I_s, load balancing indicators for routing and storage respectively, as the ratio between the maximum fraction of the load hosted by any node in the system to the average fraction of load (routing and storage) hosted by the nodes in the system. Clearly, the parameters above are low-level parameters of the architecture; however, we show in Section 4.4.1 that there is a tight relationship between these and high-level requirements such as matching latency or throughput. It can be easily proven that, in the worst case[2]: $H \cdot R \cdot I_s \geq N$ and $H \cdot R \cdot I_r \geq N$ [16]. The relations above show that optimizing some aspects of the solution (such as H and R) will lead to less scalability on the remaining dimension (load balancing).

Event delivery is the problem of dynamic multicast, where the set of destination hosts can change with every message. The difficulty here is twofold: first, the optimal topology can change with every message, and thus it should be computed accordingly; and second, even computing the multicast tree for a single message is difficult when optimizing for specific cost metrics (such as total number of hops)—in this case, the problem is related to computing the minimum Steiner tree, which is known to be NP-complete [18].

2.2 Application Heterogeneity

We selected and analyzed a few applications that either have been proposed as suitable for CBPS by other researchers or seem natural for this interaction model. The results are summarized in Table 1. The first observation is that applications suitable for CBPS-style interaction exhibit significant diversity. The wide span of input loads (with publication frequency ranging from 1 to 10^3 events per second), combined with the diversity in requirements (tolerable latency varies from 0.01s to 10s) and different requirements for expressiveness seriously complicates the task of creating a one-size-fits-all solution.

In contrast, most existing CBPS proposals are not tailored for specific applications, aiming for generality. Here, architectural design decisions (that embody various optimizations) are usually based on *expected* desirable properties of the applications rather than the specifics of a singular application. However, few applications can naturally exploit the features of any one architecture.

Let us consider an example. SIENA [4] uses a spanning tree of brokers to implement distributed CBPS. Subscriptions are propagated from the broker closest to the subscriber to all the brokers in the network, with the optimization that a subscription is not propagated if a more general subscription has been propagated already. SIENA aims to minimize bandwidth consumption and, when possible, memory utilization. An ideal application that would benefit from SIENA should exhibit the following properties: subscribers with matching interests should be connected to nearby servers (to make bandwidth and memory savings worthwhile), there should be enough commonality between subscriptions, and applications should be insensitive to latency (as content-based matching is performed at each forwarding step). Of the applications we have listed, it seems that only location-based services have the required type of subscription clustering, if subscribers only subscribe to events based on their current geographic position. However, work in this area shows that it is more natural for mobile devices to act as publishers of their current location, while content-providers subscribe (using spatial subscriptions) to areas of interest [5]. The analysis of other proposed solutions (such as Gryphon [1] and Hermes [15]) leads us to similar findings: few applications can naturally exploit the characteristics of any one existing architecture.

2.3 Lack of Wide-Area Deployment

The current status quo in CBPS, with no architecture widely deployed, delays the creation and deployment of applications that use CBPS. Closing a vicious circle, the lack of applications impairs the development of improved, real-life usable solutions.

CBPS represents a compromise between the extremes of publisher-side filtering of messages (with events directly transmitted to interested subscribers) and subscriber-side filtering of messages (with events broadcasted to all subscribers). Depending on the assumptions of the application, some solutions are more desirable than others. However, it is felt that CBPS obtains important performance bene-

[2]The worst case is when the number of subscriptions matched by a notification is linear in the total number of subscriptions and vice-versa

fits and achieves better scaling for a wide range of applications [4, 1, 15]. Nevertheless, applications that appear suited for CBPS will be implemented in other (perhaps more expensive) ways, if these solutions are readily available. Therefore, the availability of CBPS solutions on wide-area networks is a prerequisite for application development.

There is an interesting parallel to note in the very active research area of distributed hash tables (DHTs). Before 2004, there had been an impressive amount of research in the area and a large range of applications had been proposed and implemented by the DHT community. However, nobody from outside the community had built an application that used DHTs. Asserting that the reasons were the difficulty of running a DHT for a long time combined with the need for a large infrastructure, researchers proposed an infrastructure deployment of a DHT that would be publicly available. To this end, OpenDHT [19] was created and deployed on the PlanetLab testbed. Spurring the adoption of CBPS appears a more difficult problem; it is highly unlikely that an infrastructure deployment of a single particular solution to CBPS can support all possible types of applications, and therefore deploying new applications might also require deploying new solutions for CBPS.

3 Layering CBPS

Current monolithic approaches have not been adopted by real applications and have not been widely deployed yet. Although they can theoretically achieve optimal performance, it appears that creating a solution that is optimal for any given application is rather difficult with a monolithic approach. In our attempt to make the general CBPS problem tractable, we propose to solve the two sub-problems separately. This can be thought of as a layering of content-based publish/subscribe, in two sub-layers: *content-based matching* and *event delivery*.

The event delivery layer's task is to deliver messages to the subscribers and exports a single API to the content-based matching layer: *send(message, destination list)*. The content-based matching layer's task is to find the subset of subscriptions that match a given notification, which are subsequently passed to the event delivery layer. Note that this operation is distributed, as the content-based matching layer will not necessarily gather the full destination list at one single node before initiating the delivery of the event.

Layering brings a series of benefits but also has some costs. The most important drawback is the ability to independently optimize the two layers disregarding their behavior in composition, which can lead to an overall sub-optimal solution. This is not discouraging: if the resulting performance is *good-enough* for most applications, then layering is a viable choice. In a similar argument, creators of SQL argued for good-enough performance for databases that could be easily tailored for every application. Clearly, the performance of SQL-based databases is worse than that of application-tailored implementations, but this is outweighed by the advantage of easily supporting new applications or changes to existing solutions. Our argument is stronger in that application-tailored solutions seem very difficult to obtain in CBPS in the first place. Asserting that the costs are moderate, we propose a research agenda focusing on ease of reconfiguration. To this end, we propose a proof-of-concept configurable content-based matching solution in Section 4.

3.1 Benefits of Layering

The main gains are *simplicity* and *modularity*. Solving CBPS becomes now equivalent to solving its two subproblems; this endeavor, although far from trivial itself, is significantly simpler than the original problem. Modularity allows for *easier adaptability*, *differentiated scaling* and creation of *separate trust domains*.

Layering implies *easier adaptability*, as new applications can be supported by combining different solutions for content-based matching and event delivery. Several solutions to both problems already exist that could be used straightaway or slightly adapted to work in the CBPS context. Practically all the work done on supporting enhanced lookups on DHTs (such as range queries [2] or keyword searches [21]) can be used to implement content-based matching. Numerous event delivery solutions already exist (see [3], for instance) or can be adapted from existing multicast proposals.

Frequently, there is a *different scale* needed to solve the two sub-problems. For instance, stock quote subscriptions are likely to be stable and on the order of millions, if we assume that clients are individual investors. Using a cluster of well provisioned nodes (tens of nodes) that partition the subscriptions is a good solution for the content-based matching layer [22]. On the other hand, ensuring the timely delivery of the events to the numerous subscribers requires a larger number of geographically scattered nodes.

Modularity also allows a *separation of the domains of trust* in a CBPS solution, which might be mandatory for security-sensitive applications. The two layers require differentiated access to data: the matching layer *requires* access to notification and subscription payloads to perform content-based matching, as confidential content-based routing is fairly expensive [17]; in contrast, the event delivery layer need not access the subscription or notification payload, requiring instead the addresses of the subscribers. Returning to the stock quote example, subscribers' interests will only be divulged to the cluster of servers (which can potentially be trusted), while the more numerous nodes in the event delivery layer only deliver encrypted notifications. Security sensitive applications *are forced* to use layering in order to minimize the number of nodes that must be trusted with confidential information.

3.2 Costs of Layering

The costs associated to layering are due to inter-layer signalling—non negligible when the layers do not reside on the same nodes—and to the cost of maintaining partially overlapping data structures in both layers. Self optimization of both layers disregarding their behavior in composition can lead to a sub-optimal solution to CBPS.

4 Configurable architectures

We have seen that applications suitable for CBPS are quite diverse and that one-size-fits-all static solutions are impossible to create. This mandates that application requirements must be taken into account when designing CBPS solutions. Therefore, supporting a new application would necessarily trigger a lengthy process of designing a solution, implementing it and evaluating its performance.

We have proposed layering as a first step towards application-specific optimizations. To further increase the ease of supporting different applications, we advocate the creation of configurable solutions such that, given a new application, we can tune such a solution to provide acceptable performance for that application. For brevity, we will focus in more detail on configurable solutions for content-based matching, ignoring similar solutions for event delivery.

In the upcoming sections, we describe a configurable architecture for content-based matching and show that adjusting its parameters influences high-level properties such as matching latency or maximum publication throughput. The purpose of this architecture is illustrative, aiming only to show that it is possible to create configurable architectures; this simple solution has a number of deficiencies that might preclude its direct use in the real world.

4.1 Mapping Performance Requirements to Architecture Characteristics

Model. We derive a simple model of a hypothetic content-based matching architecture and study the dependency of two representative application requirements—throughput and latency—on parameters describing the architecture. We assume that brokers are connected in an overlay network (in some arbitrary topology), the links between the brokers have infinite bandwidth, and that per-link message propagation times are constant throughout the network. We define R as the average number of replicas per subscription and H the maximum number of nodes a notification visits before meeting all relevant subscriptions.

Assuming that all nodes are idle, the content-based matching latency of a message is in our simplified model: $latency = k_1 \cdot H + \sum_{i=1}^{H} match(store_i)$. In the formula, k_1 is the constant propagation time of a message and $match$ denotes the local matching time at node i as a function of the number of subscriptions stored at i, $store_i$.

The formula shows that $latency$ directly depends on H, and therefore optimizing $latency$ can be achieved by selecting proper values for H. Local matching time increases at most linearly with $store_i$ as long the node's memory size (M) is large enough to hold all subscriptions, and quickly deteriorates after the memory is full due to accesses to secondary storage. To optimize latency, we choose $H = \lfloor \frac{\sum subscriptions}{M} \rfloor + 1$.

The maximum matching throughput without loss obtained by a string of idle nodes is the throughput of the most loaded node: $throughput = \min_{i=1}^{H} \frac{1}{match(store_i)}$.

Optimizing matching throughput when $\sum stored_i$ is constant, is equivalent to evenly balancing subscriptions across the H nodes, thus ensuring that $I_s = 1$. When H increases, assuming $I_s = 1$, throughput will increase. In reality, this need not necessarily be the case, as bottleneck links and nodes might be included in the string of nodes therefore producing the opposite effect.

The calculations above assume all nodes are idle. However, local matching time experienced by a single packet depends on the load of the node; when the node becomes congested, the service time will increase sharply due to exploding queue sizes. In this case, $latency$ will increase and $throughput$ will decrease. We resort to replication to spread the notification load: notifications will follow one of multiple available sequences of H nodes to find matching subscriptions. Minimizing the maximum latency and maximizing the minimum throughput of these strings of nodes requires evenly balancing routing load, achieved when $I_r = 1$. Therefore, increasing R and minimizing I_r benefits both $latency$ and $throughput$.

Discussion. Our simplified model shows that application level requirements such as $latency$ and $throughput$ can be improved by controlling lower level parameters such as load balancing (I_r, I_s), replication rate R or number of hops H. Although the real-world dependencies of the high-level metrics cannot be inferred from this simple model, we can qualitatively extrapolate these dependencies for the general case; the only danger is that real world restrictions (such as limited bandwidth) will dampen the effect the low-level parameters have on their application level counterparts.

4.2 A Configurable Architecture

Model. Let N be the number of available nodes, R the desired replication rate and H the maximum number of routing hops. We assume all the nodes have unique identifiers chosen uniformly and randomly from a Chord-like circular identifier space (see Morris et al. [13]). The circular space is divided into X contiguous regions numbered 0 to $X - 1$, such that $N = X \cdot R$. The cluster identifier space is also circular (i.e. it wraps). The nodes (approximately R) that belong to the same part of the circular space form a cluster.

We assume that a broadcast primitive is available inside a

cluster and that clusters are connected such that every node in cluster t connects to all the nodes in cluster u, for $t \neq u$, resulting in an almost complete mesh of nodes.

Given knowledge of R and X, each node can compute the cluster it belongs to by using its identifier and estimating N from the number of entries in its routing table. Therefore, the cluster membership protocol is completely distributed.

When a subscriber wishes to express its interests, it uses a rendezvous function that takes as input the subscription and outputs an integer value t that denotes one of the X clusters. The subscriber will also select a random integer r in the range $[-\frac{H}{2}, \frac{H}{2}]$. The subscription will be sent to a random node in the cluster numbered $t+r$. The subscription will then be replicated to all the nodes in the $t + r$ cluster ($\sim R$ nodes) by using the broadcast operation.

The publication process is a random walk through H nodes, each belonging to a different cluster, with the purpose to evenly balance load across nodes in a cluster. When a message is published, the publisher will apply the rendezvous function to the notification to obtain a cluster number t. The notification will be sent to a random node in the $t - \frac{H}{2}$ cluster. This node matches it against local subscriptions and forwards it to a random node in the cluster numbered $t - \frac{H}{2} + 1$; the new node will do the same until H nodes are traversed by the notification.

Solution Properties. The maximum number of routing hops is H. The replication rate is R. The load balancing of the solution is $I_r = \frac{N}{H \cdot R}$ and $I_s = \frac{N}{H \cdot R}$. Since R and H can be chosen arbitrarily, this simple architecture is indeed easily configurable and can potentially support completely different classes of applications.

Complete connectivity. Recent work has shown that maintaining complete routing tables for overlays containing up to a few million nodes is acceptable, under reasonable rates of churn [8]. If this cost is too high, we can trade-off some routing efficiency (stretching paths with a factor of $\log N$) for smaller routing tables (either O(1) [12] or O($\log n$) [13]). In this case, N can be estimated by querying a single random node [10].

Rendezvous functions. Subscriptions and notifications are directed to corresponding clusters by using rendezvous (RV) functions. Ideally, similar subscriptions should be stored on the same cluster, to improve matching performance. The fact that we use single-output rendezvous functions (i.e. a subscription must be stored on a single cluster), limits the class of usable RV functions. One possible RV function for subscriptions would be to use a primary key attribute of the notifications to partition the subscription space. For instance, if *topic* is an attribute present in every message, partitioning its range would yield the desired RV function. Alternatives include the functions used in Hermes [15], which hash the event type of notifications and subscriptions to obtain the storage node.

Flexibility. The solution we have described can support multiple applications with different characteristics on the same infrastructure of nodes. If every application includes the same application-dependent configuration parameters (R and H) in all its messages, nodes can perform the necessary operations (random routing and cluster replication) by partitioning the virtual space for each received message. The overhead of this scheme is small.

Caveats. This solution, albeit simple and easily configurable, has some deficiencies. The average performance is the same as the worst case performance, as the solution treats every application as having uniform load. It would be desirable to have a small $average(H)$ and only bound the worst case routing hops with H. A consequence of this is that subscriptions are replicated without considering their characteristics: it would be desirable to correlate an increase in the replication rate of a subscription with an increase in the number of notifications it matches. Furthermore, all the available nodes are automatically assigned to the application, disregarding the actual load of the application; a better policy would be to provide the minimum number of nodes that supports the application's load. Finally, the measure of load balancing (for both routing and storage) is relative, making the implicit assumption that nodes are homogenous. Clearly this is not the case in a real environment, as absolute loads need to be taken into account. All of these constitute interesting problems for future research.

5 Deployment Issues

The discussion so far has focused on ways of ensuring easier adaptability of CBPS solutions to different applications. The development of new solutions, however, must be accompanied by wide-area deployments that can be used to compare the performances of different solutions and enable real applications. In this section we aim to complement the argument of this paper by describing techniques that would ease the deployment of CBPS; the technical realization of some of these proposals is a research agenda in itself.

The PlanetLab testbed [14] seems to be a good candidate infrastructure for deployment of research prototypes of CBPS solutions and applications, as its several hundred networked nodes are freely available to the scientific community. To minimize the effort of development cost and to favor cooperation, the functionality of PlanetLab nodes must be enhanced in several ways. First, *APIs* should be specified that ensure interoperability of different CBPS building blocks. A built-in *code-base* should provide functionality common to all CBPS solutions such as networking, fault-tolerance and logging. These will lower the effort of developing a new solution down to the development of its added functionality. Finally, a *library of existing solutions*

should be available to allow multiple applications to seamlessly compose a CBPS solution. This base should also be extensible with new proposals.

Beyond this basic functionality, there seem to be opportunities as well as challenges for providing more complex features, such as global load balancing and fairness to ensure "friendly" usage of PlanetLab nodes. Assigning to each application the number of nodes appropriate to its input distribution, combined with load balancing, is another research path worthy of exploration.

6 Summary

Despite its appeal and the valuable research in this area, wide-area CBPS is not currently used by any large-scale application. In this position paper, we have argued for a series of approaches to remedy this state of affairs.

We propose *layering* CBPS into content-based matching and event delivery, in an attempt to contain the overall complexity of the problem. Layering favors simplicity and modularity. Modularity allows adaptation through the composition of content-based matching and event delivery solutions, some of which are already available. Modularity also allows the use of different scales for the two layers and implicitly creates separate domains of trust that can be used to accommodate security-sensitive applications.

Observing that applications for CBPS exhibit significant diversity, we advocate the development of configurable solutions for content-based matching and event delivery that can foster diversity through reconfiguration. We show that application requirements are tightly related to architecture characteristics—such as replication rate, number of routing hops and load balancing—and propose a simple configurable architecture for content-based matching. Although not deployable in its current form, this architecture serves as a proof-of-concept that reconfiguration is possible; we hope it will spark new research in this direction.

Acknowledgement

We would like to thank Lucian Popa for numerous discussions and thoughtful reviews on previous versions of this paper. Costin Raiciu is supported by a UCL Departmental Studentship. David Rosenblum and Mark Handley hold Wolfson Research Merit Awards from the Royal Society.

References

[1] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of ICDCS*, 1999.

[2] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of SIGCOMM*, 2004.

[3] F. Cao and J. P. Singh. Medym - match early with dynamic multicast. In *Proceedings of Middleware*, 2005.

[4] 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(3), 2001.

[5] X. Chen, Y. Chen, and F. Rao. An efficient spatial publish/subscribe system for intelligent location-based services. In *Proceedings of the Workshop on Distributed Event Based Systems*, 2003.

[6] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of internet worms. *SIGOPS Oper. Syst. Rev.*, 39(5), 2005.

[7] J. Farber. Network game traffic modelling. In *Proceedings of Workshop on Network and system support for games*, 2002.

[8] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *Proceedings of Networked Systems Design and Implementation*, 2004.

[9] T. Henderson. Latency and user behaviour on a multiplayer game server. In *Proceedings of the Workshop on Networked Group Communication*, 2001.

[10] K. Horowitz and D. Malkhi. Estimating network size from local information. *Inf. Process. Lett.*, 88(5):237–243, 2003.

[11] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *Proceedings of Internet Measurement Conference*, 2005.

[12] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of Symposium on Principles of distributed computing*, 2002.

[13] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of SIGCOMM*, 2001.

[14] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1), 2003.

[15] P. R. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of the Workshop on Distributed Event Based Systems*, 2002.

[16] L. Popa, 2005. Private Communication.

[17] C. Raiciu and D. S. Rosenblum. Enabling confidentiality in content-based publish/subscribe infrastructures. Technical report, University College London, 2005.

[18] S. Ramanathan. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Trans. Netw.*, 4(4), 1996.

[19] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: A public dht service and its uses. *SIGCOMM Comput. Commun. Rev.*, 35(4), 2005.

[20] J. Risson and T. Moors. Survey of research methods towards robust peer-to-peer networks: Search methods. Technical report, University of New South Wales, 2004.

[21] C. Tang, Z. Xu, and M. Mahalingam. psearch: Information retrieval in structured overlays. *SIGCOMM Comput. Commun. Rev.*, 33(1), 2003.

[22] Y.-M. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H. J. Wang. Subscription partitioning and routing in content-based publish/subscribe networks. In *Proceeding of International Symposium on DIStributed Computing*, 2002.