A Comparative Study of Pub/Sub Methods in Structured P2P Networks

Matthias Bender, Sebastian Michel, Sebastian Parkitny, Gerhard Weikum

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
{mbender, smichel, parkitny, weikum}@mpi-inf.mpg.de

Abstract. Methods for publish/subscribe applications over P2P networks have been a research issue for a long time. Many approaches have been developed and evaluated, but typically each based on different assumptions, which makes their mutual comparison very difficult if not impossible. We identify two design patterns that can be used to implement publish/subscribe applications over structured P2P networks and provide an analytical analysis of their complexity. Based on a characterization of different real-world usage scenarios we present evidence as to which approach is preferable for certain application classes. Finally, we present simulation results that support our analysis.

1 Introduction

1.1 Motivation

Peer-to-Peer system have been a hot research topic for years now, but only recently there have been some success stories of actually deploying legal P2P-based applications on a large-scale basis, such as BitTorrent or Skype. We feel that this is partly due to the fact that the P2P paradigm has been applied to all kinds of popular (web) application scenarios, even though they mostly did not necessarily expose any natural P2P-like usage scenario in practice.

One of the few applications that naturally fit with a fully decentralized setting are publish/subscribe (pub/sub) applications. We bring forward the following two archetypical pub/sub scenarios that we will characterize and refer to throughout the paper. Consider a *User-to-user scenario* in which users want to share community knowledge, e.g., on computer troubleshooting, or discuss recent events in Blogs or Wikis. In this scenario, all users are interested in (i.e., subscribe to) a certain subset of new content, and become publishers themselves at a low rate, publishing a new article or adding a comment to an existing Blog entry. For example, imagine a publish/subscribe application where subscribers want to be notified when any participating Blog publishes a new article that contains the term P2P. In the *Publisher-to-user scenario*, a smaller set of publishers (e.g., news agencies like Reuters or AP) publish new content at a much higher rate, and a large number of users (typically distinct from the set of publishers) is interested in monitoring a developing news story, staying up-to-date on a competing business, or getting the latest tabs on a celebrity of the favorite sports team.

In spite of exposing some very distinctive characteristics, both sample scenarios expose a system model of fully distributed and autonomous information providers and information consumers, which is well-suited for a P2P-style organization. Indeed, numerous approaches on how to efficiently design publish/subscribe in a fully distributed manner have independently emerged from the P2P research community. Unfortunately, the sheer number of proposals and their often hard-to-compare assumptions regarding the underlying network structure and usage patterns render the comparison a troublesome task. Typically, the approaches are only evaluated for a very specific set of system parameters, and not compared to other existing approaches for different settings.

1.2 Contribution

This paper's contribution is threefold. First, it identifies and introduces two design patterns that can be used to implement pub/sub applications over structured P2P networks. Second, the paper provides an analytical characterization of the complexity of both approaches and provides guidelines regarding which approach might be preferable for which usage patterns. Third, we present measurements that back up both our analytical results and our suggestions for design guidelines, also in the presence of network dynamics. We explicitly model network dynamics, as peers constantly enter and leave the system. Note that this paper does not compare any concrete prototypes, but instead focuses on evaluating the design approaches for various system parameters.

The remainder of the paper is structured as follows: After Section 2 reviews related work, Section 3 introduces and discusses our system model. Section 4 introduces Store-Sub and Store-Pub, two design patterns that can be used to implement pub/sub functionality on top of a structured overlay network. Section 5 analyzes the message complexity of both approaches and subsequently gives guidelines as to which approach is well-suited for which usage scenario. Analytical results and experimental results from simulations are presented in Section 6, before Section 7 concludes this work and points at future research directions.

2 Related Work

Distributed hash tables, DHTs, (such as Chord [16], CAN [12], Pastry [13], or P-Grid [1]) have emerged as the preferred family of structured architectures for overlay P2P networks. The main advantage of DHTs compared to unstructured P2P networks stems from the performance guarantees that they can offer regarding the routing efficiency and ultimately the network scalability, even in the presence of high network dynamics (such as high rates of node arrivals/departures and failures/recoveries). P2P data networks over structured and/or unstructured networks has been a hot research topic for years [6, 9, 4, 21, 18, 11, 10].

More recently there have been numerous proposals for distributed pub/sub systems [20, 2, 7, 17, 14, 19, 8]. Most of them can be classified into three categories: topic based systems, content based systems, and hybrid solutions. Topic based systems usually consider that users subscribe to a publisher that regularly publishes documents of a certain topic (e.g. mailing lists), whereas in content based systems users subscribe to publishers that do not have a particular topic but employ a, usually, term based filtering to distinguish between relevant and non-relevant documents w.r.t. the users' interests. Hybrid systems basically consider subscriptions to topics but allow for a term based filtering. The first step reduces the communication overhead compared to the term-subscriptions case whereas the second design choice drastically reduces the number of actually shipped documents, i.e. preventing the delivery of irrelevant documents to the end user. [3] reasons on the difference between information filtering and information retrieval that can be interpreted as P2P pub/sub versus P2P retrieval.

A nice introduction into various aspects of P2P systems is given in [15].

3 System Model

We consider a network of nodes $N = \{n_1, ..., n_k\}$. Each node can take one or both of the following roles: *Subscribers* $S = \{s_1, ..., s_m\}$ express their interest in selected newly-published content, *Publishers* $P = \{p_1, ..., p_l\}$ regularly generate new content. Subscribers issue subscriptions. $sub_{i,j}$ denotes the j-th subscription of s_i, i.e., each subscriber can have more than one subscription. The average number of subscriptions per subscriber is denoted sub_{avg}. Each $sub_{i,j}$ is a set of terms t from a domain T, i.e., $sub_{i,j} \subset T$. $\overline{sub}$ denotes the average number of terms per subscription.

Publishers generate content in form of documents $d \in D \subset 2^T$, i.e., documents are a set of terms $t \in T$. Publishers issue their new content at a certain publishing rate r, measured in new documents per time interval. The notation is summarized in Table 1.

$s_i \in S$, s_i	Subscriber from set of all subscribers
$p_i \in P$	Publisher from set of all publishers
$n_i \in N$	Node in the network
$t \in T$	Term from term domain
$sub_{i,j}$	j-th subscription of s_i
sub_{avg}	average number of subscriptions per subscriber
$\overline{sub}$	average number of terms per subscription
d, $\|d\|$	document, length of document

Table 1. Notation

3.1 Discussion

For this work, we assume subscribers to express their interests in form of terms, i.e., they are interested in all new content that contains all subscription terms. We feel that more sophisticated means of subscriptions are a high burden for non-expert users, which make up a large fraction of users in a large-scale real-world network. Nevertheless, we will, at appropriate steps, present enhancements that can also accommodate more expressive subscription methods, e.g., based on XQuery expressions over RSS feeds.

Note that the ratio between publishers and subscribers and also the average number of subscriptions per subscriber highly depends on the targeted application scenario. While in the User-to-user scenario, typically all subscribing users also play the role of publishers (typically at low rates), the Publisher-to-user scenario with a limited set of news agencies as publishers exposes a small number of publishers that publish at high rates. The ratio between subscribers and publishers highly influences the optimal design pattern of the system; an extreme ratio could render either approach infeasible.

This work focuses on structured P2P architectures, such as distributed hash tables (DHTs), as a network overlay. Not limiting ourselves to a particular DHT implementation, we only assume a basic *nodeID* $\leftarrow$ *lookup(k)* functionality that maps a key k to the node identified by *nodeID* currently responsible for that key. DHTs can straightforwardly be used to construct conceptually global, but physically distributed directories. This paper focuses on structured overlays because we strongly believe that gossiping in unstructured networks inevitably leads to scalability and performance issues, in particular for application classes which aim at efficiently locating specific information such as publish/subscribe.

Our study does not include publish/subscribe approaches that let publishers circulate their documents through the whole network and that make subscribers "pick" all content that they are actually interested in, as we feel that distributing content to all peers even though they might not be interested in it exposes scalability issues already for medium-sized networks. In this context, we also point out that our analytical model and our measurements do not consider the actual document dissemination, but only the matchmaking between the subscriptions in the network and newly-published content. While we will sketch possible ways of efficiently disseminating the documents at a later stage for both design patterns, our evaluation ends as soon as the set of all matching subscribers for a new content is identified.

4 Design Patterns

The common basis for the following two design patterns to implement pub/sub functionality is the presence of a conceptually global, but physically distributed directory, built on top of all nodes in the network, i.e., subscribers and publishers alike. We use the term *directory peer* to refer to a node, stressing its participation in the distributed directory. The directory maps *keys* in form of features (e.g., terms or topics) to appropriate *values* describing the key value, e.g., in form of statistical aggregations. Such a directory can straight-forwardly be implemented on top of any DHT, offering a basic *nodeID* $\leftarrow$ *lookup(key)* method, as follows: Each node that wants to learn (or add to) the statistics for a certain key issues the corresponding *lookup(key)* request to retrieve the peer that is currently responsible for maintaining the value for that key. In a point-to-point fashion (not stressing the directory), the user can subsequently contact that peer to retrieve or add the desired information.

We strongly believe that storing (pointers to) individual *documents* in the distributed directory is not feasible, as (e.g., for Blogs in the Publisher-to-user scenario) the number of publications increases rapidly. We think it is unavoidable to abstract and aggregate the information, yielding a light-weight system that offers scalability to support an a-priori unlimited number of nodes. The following subsections describe two design patterns to implement pub/sub functionality, storing different metadata in the distributed directory.

4.1 Store-Sub

Most of the existing approaches of implementing a publish/subscribe infrastructure over structured P2P networks follow the general paradigm that we subsequently refer to as **Store-Sub**: The subscribers store their subscriptions in a conceptually global, but physically distributed directory implemented on top of the DHT. When publishers publish new content, they retrieve all applicable subscriptions from the directory, which is feasible because publishers can inverse the subscription process to find all subscriptions that match their new content. Perhaps the most obvious way to actually implement this for each $sub_{i,j}$ is to let each s_i send a message for each term t occurring in at least one of its subscription to the directory peer identified by $lookup(t)$ and to attach $sub_{i,j}$ completely. A publisher that wants to publish new content has to retrieve these lists from the appropriate directory peers for each term t of its new content in order to identify all subscribers that have issued a subscription that is matched by the content.

The bottlenecks of this approach are obvious: Publishers need to retrieve a large number of potentially huge lists of subscribers to find all appropriate subscribers when they publish new content, because they have to retrieve the subscriptions for all terms that occur in the content, in order to make sure not to miss any matching subscribers. One typical way to tackle this issue is to reduce the number of features that a publisher needs to check in order to find all appropriate subscribers. For example, this can be achieved by mapping all terms to a much smaller number of topics[1]. Subscribers send their subscription to the directory peer responsible for the appropriate topic and attach a more expressive subscription (e.g., the complete set of subscription terms or XQuery expressions). In this case, publishers only need to check for potential subscribers at a much smaller subset of directory peers.

While the most obvious way of eventually matching all such attached subscriptions might be for the publisher to retrieve the list of all subscribers for the appropriate topic(s) of a new content item and to perform the matchmaking locally, this may require to transfer an excessive number of (eventually non-matching) subscriptions from the directory peer(s) back to the publishers. Another possibility is, thus, for the publisher to transfer the actual document to the applicable directory peer(s) and let them perform the matchmaking locally, without transferring the lists of subscriptions. Subsequently, the directory peer can either start the document dissemination itself immediately or return the (much smaller) list of matching subscribers back to the publisher.

Figure 1 illustrates an example of the Store-Sub infrastructure on top of a DHT. $N17$, $N45$, and $N76$ are directory nodes responsible for topics sports, politics, comedy, and music. The mapping from topics to directory nodes is given by the DHT's *lookup()* method that, e.g., maps the topic "sport" to the directory node with nodeID 17. $N17$, for example, has already received subscriptions from $S42$ regarding the combination of terms *(worldcup, 2006, germany, opening)* and is currently receiving another subscription from $S14$. Directory peers store incoming subscriptions in a subscriber list. All publishers that generate content about the topic "sport" will turn to $N17$ to identify all matching subscribers, either by retrieving the complete lists of subscribers for all relevant topics (and subsequent local filtering) or by sending the document itself to the directory node, which will return an appropriately prefiltered subscriber list (or start the document dissemination itself).

[1] Building such a topic hierarchy is a well-addressed research problem and out of scope for this paper.

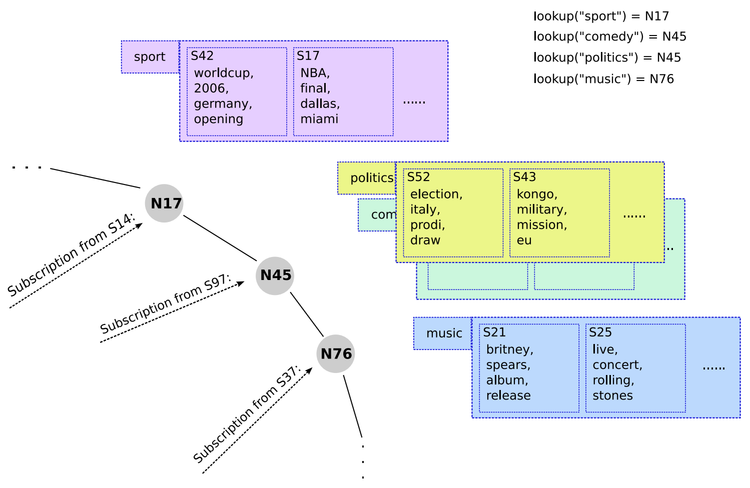

Fig. 1. Store-Sub Architecture

4.2 Store-Pub

Another approach towards implementing a publish/subscribe infrastructure over structured P2P networks is a design pattern which we will subsequently refer to as **Store-Pub**: Each publisher p_i announces its existence together with some statistical profile $prof_i$ in the distributed directory. $prof_i$ describes the content that P_i has previously published (or, potentially the expected and forecasted behavior in the near future). The distributed directory again partitions the feature (term, topic, ...) space, so that subscribers can access this (regularly updated) data to find potentially promising publishers for their information needs, and register directly with selected publisher(s).

To our knowledge, this approach has not been used so far to actually implement a publish/subscribe infrastructure. When comparing it to the Store-Sub approach, the following advantages (+) and disadvantages (-) can be identified:

+ Subscribers have full control over which publishers to contact and, e.g., prefer reputable publishers.
+ Subscribers can finetune the amount of content they receive by adapting the number of publishers they register with.
+ Subscribers do not expose information to the public that may be used for social reengineering.
+ Subscribers can subscribe to particular publishers, without being overwhelmed by content from all publishers.
- Publishers need to announce profiles. Depending on the number of publishers, that may lead to extensive network resource consumption.
- Subscribers base their decision on publishers' profiles describing the past. Subscribers may miss prospective publishers if they have not published relevant content before. If a new story arises, they cannot find any matching publishers until publishers have refreshed their profiles.

In other words, Store-Sub is an *exact* approach to pub/sub, i.e., a subscriber will in principle not miss any content that matches its subscriptions. On the other hand, Store-Pub is an *approximate* approach, there is no guarantee that a subscriber actually receives all content that matches its subscriptions, because he may not have selected the appropriate publisher based on its profile or the publisher did not yet have an appropriate profile at the time of subscription (e.g., for a developing news story). To overcome the last issue, we assume that subscribers revisit the directory on a regular basis to check whether other or better publishers have arisen to match their subscriptions, i.e., they conceptually repeat their subscription process regularly.

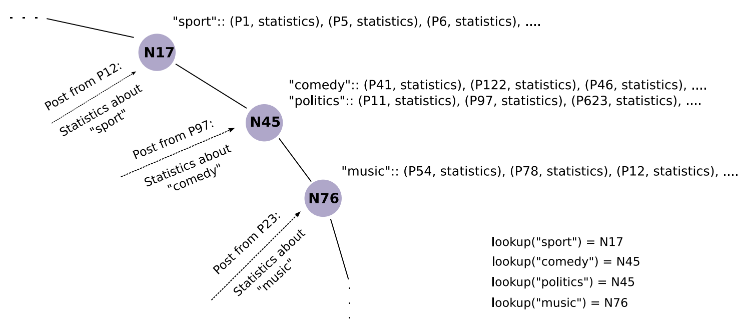
"sport":: (P1, statistics), (P5, statistics), (P6, statistics), ...
N17
Post from P12: Statistics about "sport"
"comedy":: (P41, statistics), (P122, statistics), (P46, statistics), ...
"politics":: (P11, statistics), (P97, statistics), (P623, statistics), ...
N45
Post from P97: Statistics about "comedy"
"music":: (P54, statistics), (P78, statistics), (P12, statistics), ...
N76
Post from P23: Statistics about "music"
lookup("sport") = N17
lookup("comedy") = N45
lookup("politics") = N45
lookup("music") = N76

Fig. 2. Store-Pub Architecture

Figure 2 illustrates an example of the Store-Pub infrastructure on top of a DHT. $N17$, $N45$, and $N76$ are again directory nodes responsible for topics sports, politics, comedy and music. This time, they maintain profile information describing publishers P. The mapping from features to directory nodes is given by the DHT's *lookup()* method that, e.g., maps "sport" to the directory node with nodeID 17[2]. Subscribers interested in the soccer worldcup can turn to $N17$ and identify publishers at their discretion. Registering their subscriptions is a point-to-point communication with publishers, which store the subscriptions locally.

5 Complexity Analysis

For the upcoming complexity analysis, we assume a distributed directory on top of a DHT network. As messages to an arbitrary remote node of an $|N|$-node network require an expected $log_2(|N|)$ message hops in most popular DHT architectures, we use this factor for each message sent to actually reach its destination. Regarding the notation, readers might want to refer back to Table 1.

[2] Note that it is due to the random assignment of topics/terms to peers that the statistics for "comedy" and "politics" are stored on the same peers. This does not reflect the authors' opinion ...

5.1 Store-Sub

The messaging complexity of Store-Sub consists of two ingredients:

– Subscribers joining the network need to issue their subscriptions to the distributed directory
– When publishing new content, a publisher needs to retrieve candidate subscriptions from the directory

Note again that we do not model the actual document dissemination, which is an orthogonal task as soon as all matching subscriptions are identified.

The number of messages necessary to dispatch all subscriptions of new subscribers S_{new} depends on the size of the network N, the average number of subscriptions sub_{avg}, and the number of directory nodes f_s that each subscription has to be sent to. Depending on the actual implementation, f_s can be as high as $\overline{sub}$ if the subscription needs to be sent to the directory nodes for each subscription term, or as low as 1, if the subscription only needs to be sent to a single topic directory peer (cf. Section 4.1).

The number of messages caused by publishers publishing new content depends on the number of publishers $|P|$, the *rate* at which they publish new content, the number of directory peers f_p that they need to retrieve subscriptions from (which, analogously, can be as high as the number of terms in a new document or as low as 1, if each document can be mapped to exactly one topic) and the total size of the network $|N|$. Table 2 summarizes the complexity of Store-Sub.

	Complexity
Send subscriptions (subscribers)	$O\left(\vert S_{new}\vert * sub_{avg} * f_s * \log(N)\right)$
Retrieve subscriptions (publishers)	$O\left(\vert P\vert * rate * f_p * \log(N)\right)$

Table 2. Complexity Store-Sub

5.2 Store-Pub

The messaging complexity of Store-Pub again consists of two ingredients:

– Publishers need to announce their profiles
– Subscribers need to retrieve the profiles from the directory to identify promising publishers

As the publisher's profiles can only describe previous behavior of the publishers (or, at best, forecast the future based on this previous behavior), we assume that not only new publishers P_{new} entering the system have to distribute their profiles, but also existing publishers $|P|$ have to update their profiles at regular intervals. Since the profiles are feature-(i.e., term- or topic-)specific (i.e., publishers may want to issue their profile w.r.t. each available feature), the number of messages necessary to distribute the profiles depends on the size of the feature space $|F|$, and the size of the network $|N|$.

Analogously, we expect subscribers to regularly re-check whether new publishers have become promising sources for their subscriptions, so they regularly retrieve the appropriate profiles from the directory. The messages necessary for this purpose depend on the number of subscribers, S_{new} and S, respectively, the average number of subscriptions per subscriber sub_{avg}, the number of directory nodes f_s that carry profiles relevant to a subscription, and the size of the network $|N|$. Table 3 summarizes the complexity of Store-Pub.

	Complexity
Send profiles (publishers)	$O\left((\lvert P_{new}\rvert + \frac{\lvert P\rvert}{interval}) * F * \log(N)\right)$
Retrieve profiles (subscribers)	$O\left((\lvert S_{new}\rvert + \frac{\lvert S\rvert}{interval}) * sub_{avg} * f_s * \log(N)\right)$

Table 3. Complexity Store-Pub

5.3 Discussion

For Store-Sub, which of the two contributing message types is responsible for the majority of the traffic highly depends on the system parameters. If the network volatility is high, i.e., many new subscribers S_{new} enter the system per time interval, the messages necessary to announce their subscriptions may exceed the traffic caused by the publishers generating content at low rates. Analogously, which of the two contributing message types is responsible for the majority of the traffic in Store-Pub also highly depends on the system parameters, in particular the degree of network dynamics and the average number of subscriptions per subscriber. This dependency is illustrated in Figures 3 and 4. The number of messages necessary to retrieve the subscriptions by the publishers in Store-Sub clearly dominates the messages necessary to store the subscriptions in the directory as the publishing rate increases. The number of messages to retrieve the profiles in Store-Pub dominates the number of messages to actually store the profiles if the average number of subscriptions per subscriber increases (or, analogously, if the average length of subscription increases).

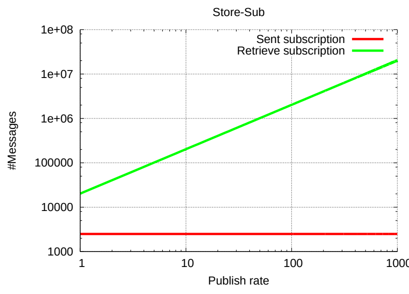

Fig. 3. Store-Sub message types

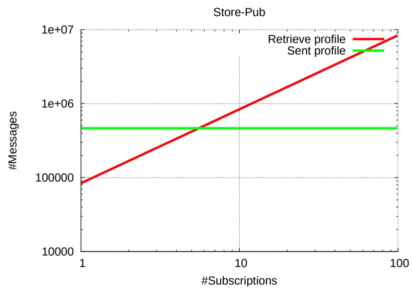

Fig. 4. Store-Pub message types

	Store-Sub	**Store-Pub**
User-to-user	+	o
Publisher-to-user	-	+

Table 4. Design Pattern Guidelines

Table 4 summarizes the analytical results by providing guidelines which design pattern is well-suited (+) or ill-suited (-) for certain usage patterns of a pub/sub system.

6 Experiments

Our experimental contribution is threefold. First, we support our analytical model of the previous section with actual simulations to verify the validity of our cost formulas. The simulation results almost exactly match the numbers forecasted by our analysis; we do not show the corresponding figures as they do not offer any insights. Second, we provide evidence based on our analytical results that, depending on the usage scenario, either one of Store-Sub or Store-Pub is the method of choice for efficiently implementing a scalable pub/sub application, as they are sensitive to different system parameters. Third, we conducted more elaborate simulations with user and document models in order to measure the actual message and traffic counts for concrete usage scenarios. The numbers back up our analysis that the resource consumption is well below reasonable limits if the implementation method of choice is in line with the expected usage pattern.

6.1 Analytical Results

For the upcoming analytical results we fix the following system parameters (cf. Table 1): $|P| = 100$, $|S| = 100,000$, $sub_{avg} = 3$, $\overline{sub} = 5$, $|T| = 100,000$, $S_{new} = 10$, $|d| = log(|T|) \sim 16$.

Figure 5 shows the sensitivity of both Store-Sub and Store-Pub to changes in the publishing rate, i.e., the amount of new content that each publisher publishes per time interval. While Store-Pub is not sensitive to this parameter, the number of messages in the Store-Sub approach increase as the publishing rate increases, as the publishers have to retrieve data from the distributed directory more often.

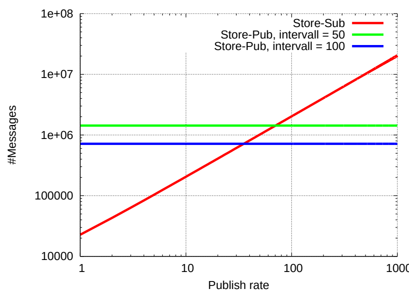

Fig. 5. Sensitivity to Publishing Rate

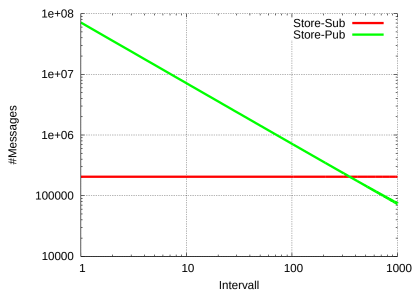

Fig. 6. Sensitivity to refresh time interval

For Figure 6 we fix the publishing rate at 10 documents per round and vary the interval at which publishers refresh their profiles in the directory and at which subscribers recheck for promising publishers. While Store-Sub is immune against these parameters, Store-Pub exposes a linear dependency. This indicates that Store-Pub is ill-suited for scenarios in which the profiles of publishers are expected to change quickly, causing the necessity to frequent profile updates.

6.2 Simulations

We have implemented a discrete event simulator mimicking the behavior of Store-Sub and Store-Pub. For this purpose, 10 publishers synthetically generate 100,000 documents using a Zipf-distribution over 100,000 terms. To achieve thematically distinct peers, we shift the terms by 20, i.e., p_i starts at t_{i*20} as its most frequent term to generate its documents. For each publisher, the first 50,000 documents were used as "seeds" to generate the publisher's profiles; the remaining 50,000 were used sequentially whenever a publisher publishes new content. The simulation starts at 5.000 subscribers ($sub_{avg} = 3$; $\overline{sub} = 5$); the average number of subscribers does not change, as expected in the Publisher-to-user scenario. At each round, a random set of 1-10 subscribers leave the system (without any notice to the system), while another 1-10 new subscribers join the system.

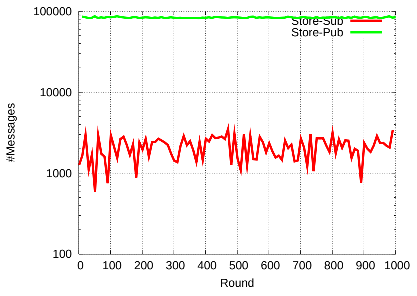

Fig. 7. Publishing rate 1 per round

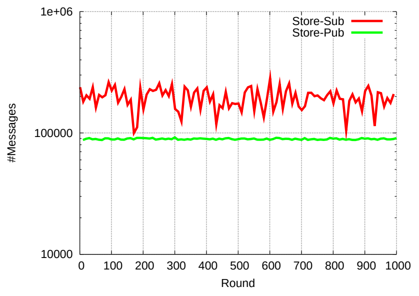

Fig. 8. Publishing rate 100 per round

Figures 7 and 8 show the total number of messages that were generated per round, where one round corresponds to one time interval (as explained for Store-Pub). It can be seen that Store-Sub generally has a larger variation in the number of messages, as the randomness introduced by new subscribers entering the system with a varying number of subscriptions is larger than for Store-Pub, where most of the traffic is introduced by the publishers refreshing their profiles (which is constant over time). Additionally, Store-Sub, as expected performs well at a lower publishing rate, while Store-Pub is immune against changes in the publishing rate.

7 Conclusion and Future Work

We have introduced two general design patterns, Store-Sub and Store-Pub, to implement pub/sub functionality on top of a structured P2P network. While Store-Sub has frequently been the basis for P2P pub/sub prototype system, we are not aware of any prototype implementing the principles of Store-Pub.

One key insight of this work is that there is no "one-fits-all" pub/sub approach, but that the optimal design pattern highly depends on a large number of system parameters, such as the expected ratio between subscribers and publishers or the rate at which publishers generate new content. While Store-Sub seems well-suited for a User-to-user scenario where publishers generate content at lower rates, Store-Pub seems attractive for a Publisher-to-user scenario where a small number of publishers generates content at high rates.

We will implement both design patterns on top of our Minerva [5] architecture. For Store-Pub, one particularly interesting field of research is how to forecast the future behavior of a publisher based on its previously published content.

References

1. K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS*, 2001.
2. I. Aekaterinidis and P. Triantafillou. Internet scale string attribute publish/subscribe data networks. In *CIKM*, 2005.
3. N. J. Belkin and W. B. Croft. Information filtering and information retrieval: Two sides of the same coin? *Commun. ACM*, 35(12), 1992.
4. M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving collection selection with overlap awareness in p2p search engines. In *SIGIR*, 2005.
5. M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Minerva: Collaborative p2p search. In *VLDB*, 2005.
6. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In *HPDC*, 2003.
7. A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-based publish/subscribe over p2p networks. In *Middleware*, 2004.
8. S. Idreos, M. Koubarakis, and C. Tryfonopoulos. P2p-diet: An extensible p2p service that unifies ad-hoc and continuous querying in super-peer networks. In *SIGMOD Conference*, 2004.
9. G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In *EDBT*, 2004.
10. J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *ECIR*, 2005.
11. I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Beyond term indexing: A p2p framework for web information retrieval. In *Informatica, Special Issue on Specialised Web Search.*, 2006.
12. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM 2001*. 2001.
13. A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.
14. A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *NGC*, 2001.
15. R. Steinmetz and K. Wehrle. *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*. Springer-Verlag New York, 2005.
16. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM 2001*.
17. D. Tam, R. Azimi, and H.-A. Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. In *DBISP2P*, 2003.
18. C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *NSDI*, 2004.
19. W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *DEBS*, 2003.
20. C. Tryfonopoulos, S. Idreos, and M. Koubarakis. Publish/subscribe functionality in ir environments using structured overlay networks. In *SIGIR*, 2005.
21. Y. Wang, L. Galanis, and D. J. de Witt. Galanx: An efficient peer-to-peer search engine system. *Available at http://www.cs.wisc.edu/ yuanwang.*