Distributed Large-Scale Information Filtering*

Christos Tryfonopoulos[1], Stratos Idreos[2],
Manolis Koubarakis[3], and Paraskevi Raftopoulou[1]

[1] University of Peloponnese, Tripoli, Greece
{trifon, praftop}@uop.gr
[2] Harvard University, Massachusetts, USA
stratos@seas.harvard.edu
[3] National and Kapodistrian University of Athens, Greece
koubarak@di.uoa.gr

Abstract. We study the problem of distributed resource sharing in peer-to-peer networks and focus on the problem of information filtering. In our setting, subscriptions and publications are specified using an expressive attribute-value representation that supports both the Boolean and Vector Space models. We use an extension of the distributed hash table Chord to organise the nodes and store user subscriptions, and utilise efficient publication protocols that keep the network traffic and latency low at filtering time. To verify our approach, we evaluate the proposed protocols experimentally using thousands of nodes, millions of user subscriptions, and two different real-life corpora. We also study three important facets of the load-balancing problem in such a scenario and present a novel algorithm that manages to distribute the load evenly among the nodes. Our results show that the designed protocols are scalable and efficient: they achieve expressive information filtering functionality with low message traffic and latency.

Keywords: publish/subscribe;distributed hash tables;information management

1 Introduction

Peer-to-peer (P2P) computing has been around for more than a decade, contributing a vast amount of research results and deployed prototypes. In this context applications that scale to millions of users and resources have been developed in domains like file-sharing (e.g., bitTorrent and Kazaa), voice and video distribution (e.g., Skype and P2PTV), distributed search engines (e.g., FAROO, YaCy, and the scientific search engine Sciencenet), and even project management (e.g., Collanos workplace).

Apart from the traditional applications discussed above, the techniques, tools and architectures introduced by the P2P paradigm have found new, interesting and useful applications also in other domains. Lately, P2P concepts are also being introduced in the –clearly different– cloud paradigm (e.g., in P2P-assisted cloud provisioning [1–3], or in hybrid architectures [4] with a backbone of super-peers

* Part of this work was performed while the authors were with the Technical University of Crete, Chania, Greece. C. Tryfonopoulos was partially supported by programme Heraclitus of the Greek Ministry of Education.

that provides access to cloud users and serves queries by executing distributed protocols), in an effort to exploit the benefits of both technologies. Additionally, P2P architectures and protocols have also been proposed in the context of distributed online social networking [5–7], aiming at solving content ownership and scalability issues, while minimising deployment and maintenance costs.

In this article, we present P2P protocols to support content and/or service lookup by utilising structured overlays, aiming at content-based filtering functionality in distributed environments.

Targeted Functionality. In the P2P architecture that we envision, resources are annotated using attribute-value pairs, where value is of type text and queried using constructs from Information Retrieval (IR) models. There are two kinds of basic functionality that we expect this architecture to offer: *information retrieval* and *information filtering (IF)*. In an IR scenario a user poses an one-time *query* and the system returns a list of pointers to matching resources. In an IF scenario, also known as *publish/subscribe (pub/sub)*, a user posts a *subscription* (or *profile* or *continuous query*) to the system to receive notifications whenever certain events of interest take place. In this article, we concentrate on the latter kind of functionality (i.e., IF) and show how to provide it by extending the distributed hash table Chord [8]. We assume that publications and subscriptions will be expressed using a well-understood attribute-value model, called $\mathcal{AWPS}$ [9]. $\mathcal{AWPS}$ is based on *named attributes* with value *free text* interpreted under the Boolean and vector space (VSM), or latent semantic indexing (LSI) models.

Our architecture and protocols target dynamic information dissemination applications, such as news alerts, digital libraries, weather monitoring, and stock quotes, consisting of large, open, and dynamic user communities. Especially for cases like news alerts and digital libraries –where the data of interest is mostly textual and users express their needs using IR languages (for example, keywords or pieces of text)– our architecture is well-suited as an implementation technology. It can handle huge amounts of information in a highly distributed, self-organising way, while offering benefits in terms of openness, scalability, and efficiency. Following our approach users, or services that act on users' behalf, would specify continuous queries for information, thus subscribing to newly appearing documents that satisfy the query conditions. The system would then notify the subscribed users automatically whenever a new matching document is published. Publishers in such a setting could be news feeds, digital libraries, or users, who post new items to blogs or other Internet communities.

Contributions. The contributions of this article are the following. We present a set of *novel protocols*, collectively called *DHTrie*, that extend the Chord protocols with pub/sub functionality assuming that publications and subscriptions are expressed in the model $\mathcal{AWPS}$. In a distributed pub/sub environment, publications typically involve contacting a large set of nodes, where matching with stored subscriptions takes place. To do this effectively, we have designed and implemented four methods that target low network traffic and low latency. In combination with these methods, we introduce a simple routing table that uses only local information and manages to significantly reduce network traffic. To justify our solution, we evaluate the DHTrie protocols experimentally in a distributed digital library scenario with hundreds of thousands of nodes and millions of user profiles. Our experiments show that the DHTrie protocols are *scalable*: the num-

ber of messages needed to publish a document and notify interested subscribers remains almost constant as the network grows, while latency is kept low. As probability distributions associated with words in publications and queries are skewed, balancing the node load becomes an important issue. We study three cases of load balancing for DHTrie, namely *query*, *routing* and *filtering* load balancing, and present a new algorithm that tackles the load balancing issues.

Preliminary results of this research have appeared in [10]. The current article revises [10] and presents the following extensions and additional contributions. We consider the issue of latency in addition to that of network traffic and identify the relevant tradeoff in our experimental evaluation. To tackle this tradeoff, we introduce two novel methods (called *hybrid* and *continuous splitting*) for resource publication. The new approaches, although very different in philosophy and design, manage to keep publication latency low while performing well in terms of network traffic. The hybrid method is a family of novel tunable alternatives that allow a per-node parameter setting aiming at adaptability. The continuous splitting method is automatic and parameterless, which makes it easy to deploy; it has proven to be efficient to many different settings and goals.

In addition to the above novel contributions, we also include more detailed descriptions of the DHTrie protocols and their respective data structures (see Section 3), and extend the experimental work (Section 4) with measurements of the new methods and comparison with the ones presented in [10], comparison under two different corpora, and experiments for publication latency and network dynamics. Finally, we redesign and apply the algorithm presented in [10] to query load balancing, and study its effects on message traffic (Section 4.9).

Organisation. The organisation of the article is as follows. Section 2 positions our work with respect to related research, while Section 3 presents the DHTrie protocols. The experimental evaluation of DHTrie and a study of load balancing issues are presented in Section 4, followed by Section 5 that concludes the article.

2 Related Work and Background

In this section, we survey related work in the area of pub/sub and IF in P2P networks. Naturally, this paper is also relevant with the broad area of distributed query processing, with studies on different query models in distributed settings (i.e., one-time, relational, and RDF query processing), and with the area of IR in P2P networks as it shares many common goals and techniques with IF.

2.1 P2P Pub/Sub and Information Filtering

Work on pub/sub in distributed systems has contributed some fundamental ideas that have also been utilised in the P2P domain. Researchers in this area have developed various data models based on channels, topics, and attribute-value pairs to represent publications and subscriptions. Pub/sub systems based on attribute-value models are called *content-based*, as attribute-value data models are flexible enough to express the content of publications. The query languages of content-based pub/sub systems are based on Boolean combinations of arithmetic and string operations. Work in this area has concentrated not only on distributed pub/sub architectures, but also on filtering protocols.

SIENA [11] is probably the most elegant example of a system to be developed in this area. A very important contribution of SIENA is the adoption of a

P2P model of interaction among servers and the exploitation of traditional network algorithms based on shortest paths and minimum-weight spanning trees for routing messages. The core ideas of SIENA have been used in the early P2P pub/sub system P2P-DIET [12].

With the advent of DHTs such as CAN, Chord, and Pastry a new wave of pub/sub systems has appeared. Scribe [13] is a topic-based pub/sub system based on Pastry. Hermes [14] is similar to Scribe since it uses the same underlying DHT but it allows more expressive subscriptions by supporting the notion of an event type with attributes. Related ideas appear later in [15, 16] and in PeerCQ [17], a notable pub/sub system implemented on top of a DHT infrastructure designed to cope with peer heterogeneity by extending consistent hashing [18].

The study in [19] is mainly concerned with scalability of current designs and proposes two methods that allow to restrict the overall costs. Both these methods can improve general purpose P2P protocols and can be applied on top of our work as well. [20] study the problem of content-based pub/sub functionality on top of Chord, allowing for range-based subscriptions, i.e., one can define a range for a given attribute as opposed to a single value. Such ideas can readily be adopted by our protocols as well. Meghdoot [21] is another pub/sub proposal in the area of DHTs that uses ideas such as hashing of subscriptions and events to facilitate matching. The difference of Meghdoot from our work is that it is built on top of CAN, whose characteristics are heavily exploited in the system design (e.g., it uses zone splitting/replication).

Research on processing subscriptions using string attributes in DHT-based pub/sub systems is also related to our work. PastryStrings [22] utilises prefix-based routing to facilitate processing of publications that are strings, and subscriptions that are string predicates. Additionally, the DHTStrings system [23] utilises a DHT-agnostic architecture to support prefix and suffix queries in string attributes. More recent works on P2P pub/sub systems have focused on various issues such as new routing protocols [24–26], combination of IR and IF [27], web services [28], load balancing [29], security [30] and preference awareness [31].

Similarly to the pub/sub strand of research, approaches that use a DHT as the routing infrastructure to build filtering functionality for IR-based models and languages have also been introduced. Closer to our work are the systems pFilter [32] and Ferry [33].The main qualitative difference of our work is that we support a different and more expressive query model, requiring more complex protocols. In addition, from a quantitative point of view our work provides a more in depth analysis by stressing the system to millions of queries and tens of thousands of nodes as opposed to only thousands of queries and thousands of nodes in [32, 33]. Below we discuss these works in more detail, and compare them against our approach.

pFilter [32] is the closest system to the ideas presented in this work. It uses a hierarchical extension of CAN [34] to store user queries and relies on multi-cast trees to notify subscribers. Compared to pFilter, our work uses a more expressive data model and query language, while there is no need to maintain multi-cast trees to notify subscribers. However, the multi-cast trees of pFilter take into account physical network distance something that we do not consider at all in this work, but rather we consider publication latency and load balancing issues.

Ferry [33] is another proposal to support IF functionality on top of DHTs. The main difference of our work is the support of a more expressive and complex data and query model. The main novelty of Ferry is that it exploits the DHT links, e.g., the contents of the Chord finger table, to disseminate information in the network. In our work, we exploit similar ideas by extensively taking advantage of DHT links, trying to group messages based on the Chord finger table, and piggy-backing information on maintenance messages. In addition, we provide further routing flexibility with the addition of a routing table, called FCache (see Section 3.5), that consists of a low cost and best effort cache of IP addresses that allows us to bypass the DHT protocol whenever this is possible. Essentially, this comes at zero network cost as it is a process piggy-backed on the normal DHT messages.

2.2 Other Related Areas

Distributed query processing relies on distributed protocols that dictate where data meets queries. Depending on the network design and properties, and on the query model utilised, different query processing algorithms are needed. The first systems to cope with distributed query processing were mainly based on strictly structured designs and focused on relational query processing [35].

Mariposa [36], one of the most well-known distributed database systems and probably the most ambitious attempt to scale to thousands of nodes, proposes node interaction protocols based on economic models. Another well known distributed database system is LH* [37], where the authors introduce the notion of the scalable distributed data structure (SDDS).

The *early P2P designs* in the area tried to remove all the restrictions of the classic distributed systems. However, the more demanding nature of applications enforced structure in P2P networks, appearing in the form of DHTs and hypercubes [38]. Essentially, these architectures provide functionality so as data items or queries can be *mapped* to certain node(s) given a set of properties and functions. In this way, structured networks provide a non-centralised but still controllable design pattern.

These network designs can be seen as a hybrid between the early distributed systems and the early P2P networks, trying to balance the various tradeoffs and thus, offering extensive flexibility and adaptability to build any kind of application over it. Thus, there has been a tremendous amount of research over structured P2P networks, e.g., there is work on relational one-time and continuous query processing [39–41, 17, 42, 43] and RDF query processing [44–47].

Information retrieval is the dual problem of information filtering, often referred to the other side of the same coin [48]. Although many of the underlying issues are similar as both IR and IF share the common goal of information delivery to information seekers, the design issues (e.g., timeliness of data, identification and representation of user needs), and also the techniques and protocols to satisfy these information needs differ significantly.

In [40], one of the early works that considered how to process IR queries on top of DHTs, the authors discuss issues involved in building IR functionality over text databases on top of structured overlays. In a similar spirit, [49] discusses the feasibility of Web search in a P2P environment and estimates the difficulty of the problem. A straightforward approach to support Boolean searching in

P2P networks is presented in [50], where each node in the network is responsible for a specific keyword through the DHT hash function and the focus is put on multiple keyword queries.

Many works have studied how to support document querying based on VSM on top of structured overlays. Meteorograph [51], one of the early works that deal with similarity search over structured P2P networks, describes how to support similarity and ranked search in a linear hash addressing space overlay. In another approach, LibraRing [52] proposes a two-tier architecture for a digital library environment aiming to unify IR and IF in a single framework. While most of related papers utilise a DHT to route the queries to appropriate peers, in Minerva [53] a global distributed directory for IR-style statistics and quality of service information is built at indexing time, to be then exploited at query time.

Finally, *cloud/grid computing* and *social networking* have emerged over the last couple of years as new paradigms and application areas for distributed data management. Our research is also related to works in this domain, as researchers exploit and extend ideas from the distributed/P2P domain to provide new data management functionality as in [1, 5, 6, 54, 55].

3 The DHTrie Protocols

We implement pub/sub functionality by a set of protocols called *DHTrie* (from the words DHT and trie). The DHTrie protocols use *two levels of indexing* to store submitted queries.

The first level corresponds to the partitioning of the global query index to different nodes using DHTs as the underlying infrastructure. Each node is responsible for a fraction of the submitted queries through a mapping of attribute values to node identifiers. The DHT infrastructure is used to define the mapping scheme and also manages the routing of messages between different nodes. We use an extension of the Chord DHT [56] to implement our network. The set of protocols that regulate node interactions are described in the next sections.

The second level of our indexing mechanism is managed locally by each node and is used for indexing the user queries the node is responsible for. Each node uses a trie-like data structure to perform query clustering and improve filtering performance. The details of local indexing are presented in [57].

3.1 The Subscription Protocol

Let us assume that a node P wants to subscribe with a query q composed as a conjunction of atomic queries:

$$\begin{array}{c} A_1 = s_1 \ \wedge \ ... \ \wedge \ A_m = s_m \ \wedge \\ A_{m+1} \sqsupseteq wp_{m+1} \ \wedge \ ... \ \wedge A_n \sqsupseteq wp_n \ \wedge \\ A_{n+1} \sim_{a_{n+1}} s_{n+1} \ \wedge \ ... \ \wedge \ A_k \sim_{a_k} s_k \end{array} \quad (1)$$

where A_i is an attribute, s_i is a text value, wp_i is a conjunction of words and *proximity formulas*[4], and a_i is a *similarity threshold*, i.e., a real number in the

[4] A proximity formula is an expression of the form $w_1 \prec_{\xi_1} \cdots \prec_{\xi_k} w_k$, where w_i is a word and ξ_i is a distance interval of the form $\{[l, u]\colon l, u \in \mathbb{N}, l \geq 0 \text{ and } l \leq u\} \cup \{[l, \infty)\colon l \in \mathbb{N} \text{ and } l \geq 0\}$. The proximity operator $\prec_\xi$ is used to capture the concepts of *order* and *distance* between words in a text document using intervals that impose lower and upper bounds on distances between words.

interval $[0, 1]$. For a query q of the above form, the atomic queries with equality ($=$) and containment ($\sqsupseteq$) operators will be called its *Boolean part*, while the atomic queries with similarity ($\sim$) operators will be called its *vector space part*.

To perform the subscription, P randomly selects a single word w contained in any of the text values $s_1, \ldots, s_m$ or word patterns $wp_{m+1}, \ldots, wp_n$ and computes $H(w)$, where $H()$ is a consistent hash function used to map identifiers in the identifier circle of Chord [56], to obtain the identifier of the node that will be responsible for query q. Then P creates message FwdQuery$(id(P), ip(P), qid(q), q)$, where $id(P)$ is the identifier of node P computed by hashing a piece of information that identifies P (e.g., its IP address and port, or a unique identifier given to it the first time it joins the network), $ip(P)$ is the IP address of P, and $qid(q)$ is a unique query identifier assigned to q by P. This message is then forwarded in $O(logN)$ steps to the node with identifier $H(w)$. Since only one node has to be contacted, this forwarding is done using the Chord *lookup*() function to locate *successor*$(H(w))$, i.e., the first node which is equal or follows $H(w)$ clockwise in the identifier space and is called the *successor* node of identifier $H(w)$. Once *successor*$(H(w))$ is located, it is directly contacted by P. In this way, queries of this type are always indexed under their Boolean part to save message traffic, since they need to be stored at a single node. Notice also that both $id(P)$ and $ip(P)$ need to be sent to the node that will store the query to facilitate notification delivery.

When P wants to submit a query q of the form $A_{n+1} \sim_{a_1} s_1 \wedge ... \wedge A_n \sim_{a_n} s_n$ (i.e., with a VSM part only), it sends q to *all* nodes in the list $L = \{H(w_j) : w_j \in D_1 \cup \cdots \cup D_n\}$, where $D_1, \ldots, D_n$ are the sets of *distinct* words in text values $s_1, \ldots, s_n$. In contrast to queries with a Boolean part described above, queries with *only* a VSM part need to be stored in all the nodes involved (computed as above) in order to ensure correctness of the filtering process. Sending the same message to more than one recipients is discussed in detail in the next section, where the same problem is posed again by the publication forwarding process.

When a node P' receives a message FwdQuery containing q, it stores q using the second level of our indexing mechanism. P' uses a hash table to index all the atomic queries of q using as key the attributes $A_1, \ldots, A_k$. To index each atomic query, three different data structures are also used: (i) a hash table for text values $s_1, \ldots, s_m$, (ii) a trie-like structure that exploits common words in word patterns $wp_{m+1}, \ldots, wp_n$, and (iii) an inverted index for the most "significant" words in text values $s_{n+1}, \ldots, s_k$. P' utilises these data structures at filtering time to find quickly all queries q that match an incoming publication p. This is done using a method that combines algorithms BestFitTrie [57] and SQI [58]. The details of local storage and indexing using BestFitTrie are discussed thoroughly in [57].

3.2 The Publication Protocol

Publication of a resource involves sending the same message to a group of nodes that is not known a priori. To tackle this problem, we have designed and implemented four methods: (i) the iterative method, which is the standard way to contact a number of different nodes over Chord, (ii) the recursive method, which creates a single message with all the recipients contained in a sorted list and works its way around the identifier space until all recipients have been contacted, (iii) the hybrid method which uses machinery from the two previous

methods to provide a tunable alternative between the two extremes, and (iv) the continuous splitting method, which exploits the finger tables of all message recipients to split the message at every forwarding node, aiming at the optimisation of network traffic and latency.

The publication protocol essentially involves sending the same message to the group of nodes that are responsible for the distinct words contained in the text values of the different attributes of p. In this way, when a node P wants to publish a resource, it first constructs a publication of the form $p = \{(A_1, s_1), (A_2, s_2), \ldots, (A_n, s_n)\}$ (i.e., a set of attribute-value pairs (A_i, s_i), where A_i is a *named attribute*, s_i is a *text* value, and all attributes are *distinct*) that is the resource description. Let $D_1, \ldots, D_n$ be the sets of *distinct* words in $s_1, \ldots, s_n$. Then, publication p has to be propagated to *all* nodes with identifiers in the list $L = \{H(w_j) : \ w_j \in D_1 \cup \cdots \cup D_n\}$. The subscription protocol guarantees that L is a superset of the set of identifiers responsible for queries that match p. To propagate publication p in the DHT, P removes duplicates from L and sorts it in ascending order clockwise starting from $id(P)$. In this way, we obtain at most as many identifiers as the distinct words in $D_1 \cup \cdots \cup D_n$, since a node may be responsible for more than one of the words contained in the document.

3.3 Methods for Subscription and Publication

In this section, we describe four different methods to implement the subscription and publication protocols and present their advantages and disadvantages.

The Iterative Method. Each node P, that uses the iterative method (It) to contact the recipients in list L, constructs a FwdResource$(id(P), pid(p), p, id(P'))$ message for each identifier $id(P')$ contained in L, where $pid(p)$ is a unique metadata identifier assigned to publication p by node P. Then, it utilises the *lookup*() procedure provided by Chord to locate node P' and sends it the FwdResource message. This is repeated for all the identifiers in L in an *iterative* way. Using this method, P needs $O(h \log N)$ messages, where h is the number of different nodes to be contacted. Figure 1 illustrates graphically the publication of a resource to three recipients under Chord using the iterative method and shows a message graph for a general case of resource publication under It.

The Recursive Method. Using the iterative method has an obvious disadvantage; the same node may participate in many *lookup*() requests for nodes responsible for identifiers in list L causing increased network traffic. This is the reason for designing the recursive method (Re). The idea behind method Re is to pack messages together to reduce network traffic as follows.

Having obtained L, P creates a message FwdResource$(id(P), pid(p), p, L)$, where $pid(p)$ is a unique metadata identifier assigned to p by P, and sends it to node with identifier equal to $head(L)$ (the first element of L). This forwarding is done by the following *recursive* way: message FwdResource is sent to a node P', where $id(P')$ is the greatest identifier contained in the finger table of P, for which $id(P') \leq head(L)$ holds.

Upon reception of a message FwdResource by a node P', $head(L)$ is checked. If $id(P') < head(L)$ then P' just forwards the message as described

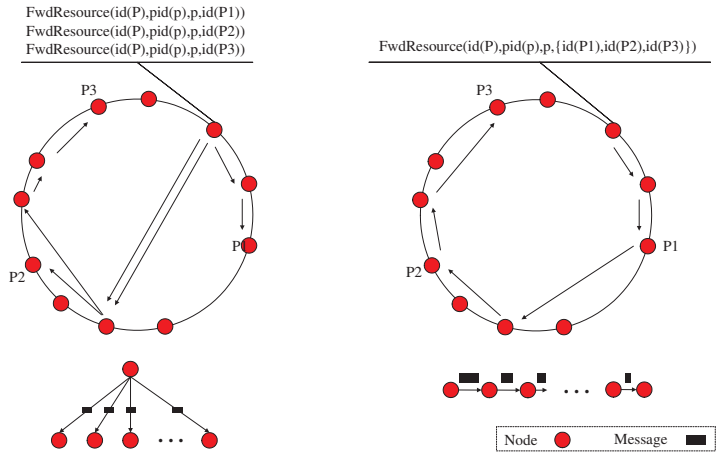


Fig. 1. Message routing and message graph for the It (left) and Re (right) methods

in the previous paragraph. If $id(P') \geq head(L)$ then P' makes a copy of the message, since this means that P' is one of the intended recipients contained in list L (in other words P' is responsible for key $head(L)$). Subsequently, the publication part of this message is matched with the node's local query database using the algorithms described in detail in [57] and the appropriate subscribers are notified. Additionally, P' modifies list L to L' by deleting all elements of L that are smaller than $id(P')$ starting from $head(L)$, since all these elements have P' as their intended recipient. For the new list L', $id(P') < head(L')$ holds. Finally, P' forwards the message to node with identifier $head(L')$. Figure 1 illustrates graphically the publication of a resource to three recipients under Chord using the recursive method and shows a message graph for a general case of resource publication under Re.

The Hybrid Methods. The idea behind the recursive method is to pack messages together to reduce network traffic. This however, comes at the cost of high latency; if the recipients list is long then the last recipient has to wait for a long time until it is notified about the publication, which in turn causes delays in the notification of the interested subscribers. The iterative method on the other hand, tries to optimise latency since no recipients lists are used and the delay to deliver a message is logarithmic in the size of the network. This of course comes at the price of high network traffic.

To tackle this tradeoff, we designed and implemented a family of *hybrid* approaches that combine the benefits of the two previous methods. The idea behind the hybrid methods is to design tunable alternatives that will provide fast delivery of messages at low network cost. To achieve this, the message originator splits the initial recipients list to smaller ones and each recipients list is sent in an iterative way, while the message is forwarded in the network recursively. The family of hybrid methods is designed to provide variations with different objectives, while the difference between the three variants presented below lies in the initial splitting of the recipients list. Finally, notice that the parameters in the hybrid methods are not global, but may be set in a per-node fashion, thus adapting to node specifics regarding publication size.

The *fixHy* method. The fixed hybrid ($fixHy$) method requires fixing a value for the *desired recipients list size* σ. Parameter setting in the $fixHy$ method, although adhoc, allows the manual tuning of the system according to document and vocabulary size, but requires expertise in setting this value. Notice that, if the average document length published in the system is changed, the method may create too short or even useless recipients lists. The $fixHy$ method works as follows.

Having obtained L, node P uses it to create $h = \lceil |L|/\sigma \rceil$ recipients lists, of size σ. In our experiments, we used $\sigma = 10$ and $\sigma = 50$ as baseline values depending on the tested corpus, and showed the effect of the desired recipients list size in the message traffic and latency observed in the network. Notice that the $fixHy$ method will degenerate to the recursive method for $\sigma = |L|$ and to the iterative method for $\sigma = 1$. Thus, using a high value for σ will make the protocol behave similarly to the recursive method, while using a low value for σ will make the protocol behave similarly to the iterative method.

The *perHy* method. The percentage hybrid ($perHy$) method requires the tuning of parameter π, which controls the percentage of the initial recipients list that will be used to create each new list. This method is less flexible than $fixHy$ in setting the size of the recipients list, but requires less expertise and is adaptable to changes in the document size published in the network. Setting the recipient list size as a percentage of the initial recipients list allows coping with both large and small documents, whereas in the $fixHy$ method this is not possible. The $perHy$ method works as follows.

Having obtained L, node P uses it to create $h = \lceil 1/\pi \rceil$ recipients lists of size $|L| * \pi$, where $0 < \pi \leq 1$. In our experiments, we used $\pi = 4\%$ and also showed the effect of π in the message traffic and latency observed in the network. Notice that the $perHy$ method will degenerate to the recursive method for $\pi = 1$ and to the iterative method for very small values of π.

The *medHy* method. The median hybrid ($medHy$) method is an automatic method that requires no parameter tuning, since the recipients lists are split according to the median of the differences between consecutive intended recipients. This method identifies the large "gaps" in the intended recipients list and splits it accordingly. Since no parameter setting is required, this is a method best suited for general purpose applications with published documents of varying size; no expertise is needed, since the recipients list is split according to its special characteristics. The $medHy$ method works as follows.

Having obtained $L = \{l_1, l_2, \ldots, l_{|L|}\}$, node P traverses it starting at $head(L) = l_1$ and calculates all differences $\delta_i = l_i - l_{i+1}, 1 \leq i \leq |L|-1$, between consecutive intended recipients in L. Subsequently, P calculates the median δ_{med} of these differences and uses it to split L in the following way. P traverses L once more starting at $head(L)$, and when $\delta_k > \delta_{med}, 1 \leq k \leq |L| - 1$, it creates a new list $L_1 = \{l_1, \ldots, l_k\}$. Subsequently, L becomes $L \setminus L_1$, while element l_{k+1} is now $head(L)$, and the process continues until L is empty. Notice that there is no way to tune this method to behave similarly to either the iterative or the recursive method, since the splitting of the initial intended recipients list is done automatically according to the keys in L. The $medHy$ method provides an automatic way to utilise the hybrid protocol, without the need for performance tuning, or any knowledge of the underlying document properties.

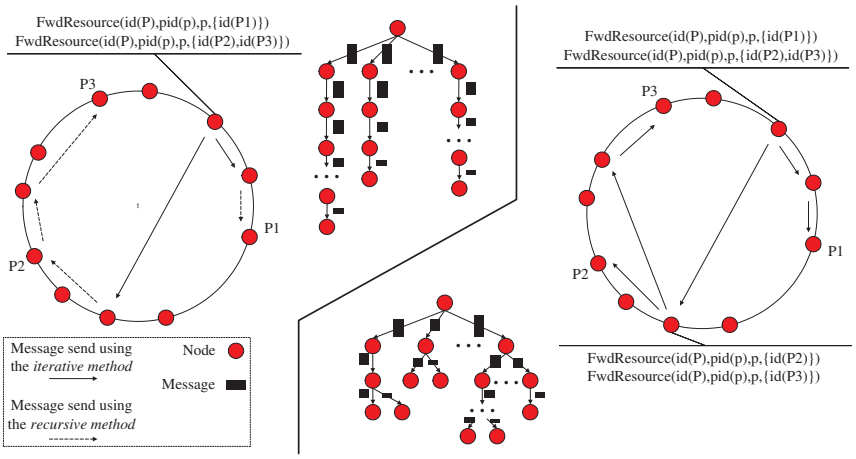


Fig. 2. Message routing and message graph for the $fixHy/perHy/medHy$ (left) and Spl (right) methods

For each one of the lists $L_1, \ldots, L_h$ created by any of the variations ($fixHy$, $perHy$, $medHy$) of the hybrid method presented above, a message of the form FWDRESOURCE$(id(P), pid(p), p, L_i)$, with $1 \leq i \leq h$, is constructed and is iteratively sent to $head(L_i)$. Since each message contains a list of recipients, the recursive method is utilised to forward the message to the rest of the nodes in list L_i. When a node P' receives a FWDRESOURCE message, it removes all elements in L that have P' as their intended recipient and forwards the message in a recursive way. Notice that only the message originator may split the message into smaller lists, while the rest of the nodes receiving it are responsible just for forwarding it. The usage of many recipients lists with smaller size together with the iterative way of sending these lists justifies the hybrid nature of the protocol. As we will show in Section 4, this method manages to achieve lower latencies than the recursive method while keeping message traffic relatively low. Figure 2 illustrates graphically the publication of a resource to three recipients under Chord using any of the hybrid methods and shows a message graph for a general case of resource publication under $fixHy$, $perHy$, or $medHy$.

The Continuous Splitting Method. All the variations of the hybrid method split the initial recipients list only once at the message originator and then, all the subsequent recipients of the messages are forced to perform the routing task based on these lists. This makes the protocol simpler, but also adds inefficiencies, since the recipients of the messages are not allowed to optimise the routing by splitting the message further. The continuous splitting (Spl) method overcomes this limitation by allowing each message recipient to split the message into sublists, according to information in its finger table. In this way, the message is split several times at each recipient before it is forwarded to other intended recipients, causing an adaptive execution of the forwarding process.

The Spl method tries to exploit each node's view of the network (in contrast to the hybrid methods that exploit only the originators' view), by splitting the intended recipients lists according to finger table entries of all the nodes participating at the forwarding of the message. The drawbacks of this method include

the need to access the finger table of each node, which may result in poor splitting of the intended recipients list if the finger table entries are outdated, or the message is too small. The *Spl* method works as follows.

Each finger table entry $f_P[i]$ in the finger table of P is used to create one or more lists in the following way. Consider two consecutive entries in the finger table of P, say $f_P[i]$ and $f_P[i+1]$. Starting from the identifier $id(f_P[i])$ stored at this entry, P scans $L = \{l_1, l_2, \ldots, l_{|L|}\}$, and collects all the recipients with identifier greater than $id(f_P[i])$ and smaller than $id(f_P[i+1])$ to create list $L_1 = \{l_1, \ldots, l_k\}, 1 \leq k \leq |L|-1$. Subsequently, L becomes $L \setminus L_1$, while element l_{k+1} is now $head(L)$ and the process continues for all the intended recipients in the list L, until L is empty. Typically, finger entries with higher index number have longer lists associated with them (remember that entries in the finger table of a node point to exponentially increasing distances away from the node), which means that typically the distance between the identifiers of entries $f_P[i-1]$ and $f_P[i]$ is shorter than the distance between those of entries $f_P[i]$ and $f_P[i+1]$.

For each one of the lists $L_1, \ldots, L_h$ created by the continuous splitting method, a message FwdResource$(id(P), pid(p), p, L_i)$, with $1 \leq i \leq h$, is constructed and is iteratively sent to $head(L_i)$. When a node P' receives a FwdResource message, it removes all elements in L that have P' as their intended recipient and repeats the procedure described above to split list L_i further according to its own finger table. As we will show in Section 4, this method manages to achieve latency as low as that of the iterative method while keeping message traffic low. Figure 2 illustrates graphically the publication of a resource to three recipients under Chord using the continuous splitting method and shows a message graph for a general case of resource publication under *Spl*.

The reader may have noticed that the publication (and also the subscription) protocol in all the proposed methods indexes queries that consist of a single equality of the form $A = s$ using a single word contained in the text value s, contrary to the standard way that would index the entire text value s in the DHT. This is done to avoid sending extra network messages for each publication to discover matching equalities. False positives that may occur are resolved locally at each node, thus relieving the network of significant message overhead.

Independently of [10], where we originally presented the iterative and recursive methods, the technical report [59] presented an approach that shares ideas with these two methods by discussing how to implement multicast functionality at different levels of a DHT architecture. However, [59] aimed at multicast from a physical network viewpoint and focused on the comparison of these techniques across the CAN and Chord DHTs.

3.4 The Notification Protocol

When a message FwdResource containing a publication p arrives at a node P, the queries matching p are found by utilising its local index structures and using the algorithms described in detail in [57] for queries with a Boolean part only. The extension to $\mathcal{AWPS}$ queries involves the calculation of the cosine of the angle of two vectors corresponding to text values from a publication and a query, and follows straight-forward IR techniques.

Once all the matching queries have been retrieved from the database, P creates notification messages of the form Notification$(ip(P), pid(p), qid(q))$,

where P is the provider that published the matching resource, and sends them to all the nodes that their queries were matched against p using their IP addresses associated with the query they submitted. If a node P' is not online when P tries to notify it about the published resource, the notification message is sent to the $successor(P')$. In this way P' will be notified the next time it logs on the network. To utilise the network in a more efficient way, notifications can also be batched and sent to the subscribers when traffic is expected to be low.

3.5 Frequency Cache

In this section, we introduce an additional routing table that is maintained in each node. This table, called *frequency cache (FCache)*, is used to reduce the cost of publishing a resource. Using the protocols described earlier, each node is responsible for handling queries that contain a specific word. When a resource r with h distinct words is published by node P, P needs to contact at most h other nodes which will match the incoming resource against their local query databases. This procedure costs $O(h \log N)$ messages for each resource published at P. Since some of the words will be used more often at published resources, it is useful to store the IP addresses of the nodes that are responsible for queries containing these words. This allows P to reach in a single hop the nodes that are contacted more often (proxying).

Specifically, FCache is a hash table used to associate each word that appears in a published document with a node's IP address. It uses a word w as a key and each FCache entry is a data structure that holds an IP address. Thus, whenever P needs to contact another node P' that is responsible for queries containing w, it searches its FCache. If FCache contains an entry for w, P can directly contact P' using the IP stored in its FCache. If w is not contained in FCache, P uses the standard DHT lookup protocol to locate P' and stores contact information in FCache for further reference. Using FCache, the cost of processing a published resource p is reduced to $O(v + (h - v) \log N)$, where v is the number of words of p contained in FCache. Notice that the construction and maintenance of FCache comes at no extra message cost and node routing information is discovered only when needed. In the experiments presented in the next section we discuss good choices for FCache size (see Section 4.4).

The only extra cost involved with FCache is due to possible cache misses because of network dynamicity. In an FCache miss, the node needs to utilise the routing infrastructure at the cost of $O(logN)$ messages to locate a node. However, the new contact information is used to update the FCache entry for future reference. Misses are most likely to occur for infrequent words, since nodes responsible for storing queries with frequent words will be contacted repeatedly.

3.6 Network Dynamicity and Fault Tolerance

The issues introduced by the dynamic nature of P2P systems may be distinguished in two general categories: (i) topology changes as nodes move in and out of the system and (ii) content changes as users shift their interests to new topics while losing interest in others.

In a dynamic network, nodes may join, leave, or fail at any time (referred to as *node churn* in the literature). The main challenge in dealing with these

situations in a DHT is preserving the ability to locate every key in the network. The stabilisation protocol provided by Chord aggressively maintains the finger tables of all nodes as the network evolves, by relying on successor pointers to undertake correctness of lookups and finger table repairs. This stabilisation scheme guarantees to offer reachability of existing nodes even at the face of concurrent joins, leaves, or fails and allows lookups to be both fast and correct. Since all nodes are uniquely identified in the network, and the Chord identifier calculated is the same for each reconnection, a node is naturally mapped at the same location on the Chord ring every time. This is exploited by the DHTrie protocols to store notifications for a node at its successor and to deliver them upon node reconnection. Naturally, successor nodes are also used for data handover when a node departs normally from the network. To cope with data loss due to node failures and accelerate lookups further, replication [60–62] and caching [63, 64] algorithms may be utilised. Finally, note that changes in network topology will also lead to FCache misses (remember that misses do not affect the correctness of the protocols) and hence, increase message traffic, as shown in Section 4.8.

Naturally, the interests of the nodes evolve over time resulting in creating, modifying, or deleting queries from the network, or even changing the topic and rate of their publications. These changes will cause an increase in message traffic as long as the network tries to cope with the content shift. Newly introduced topics or topics that have suddenly gained increasing interest will introduce new terms, which in turn will be infrequent at the beginning, but their frequency of occurrence will increase with user publications. These changes in content are expected to initially generate FCache misses (i.e., increase network traffic), but as specific terms become popular FCache will be gradually updated.

4 Experimental Evaluation

To carry out the experimental evaluation of the protocols described in the previous section, we needed metadata for incoming resources, as well as user queries. For the model $\mathcal{AWPS}$ considered in this work there are various document sources that one could consider: TREC corpora, metadata for papers on various publisher Web sites (e.g., ACM or IEEE), electronic newspaper articles, articles from news alerts on the Web (`http://www.cnn.com/EMAIL`), and others. However, it is rather difficult to find user queries except by obtaining proprietary data (e.g., from CNN's news or Springer's journal alert system). Additionally, notice that using query logs of one-time queries as continuous queries does not create realistic query databases. One-time queries are in general short and focused, as they express one-time information needs, while continuous queries tend to be longer, more complex and more general, in order to satisfy long-term information needs.

4.1 Experimental Setup

In this section, we describe the document and query sets used to evaluate our methods, and present the performance criteria and setup of our evaluation.

Document Corpora. For our experiments, we used two sets of real-life documents and queries. The first set is composed of 10,426 documents downloaded from CiteSeer (`http://citeseer.ist.psu.edu`), originally compiled in [65], and used also in [57, 52, 10]. These documents are research papers in the area of

Description	NN corpus	DBP corpus
Collection size compressed (uncompressed)	99.6 MB (346.2 GB)	548.8 MB (2 GB)
Number of documents	10,426	3,144,265
Document vocabulary size in words	379,484	2,902,491
Maximum document size in words (KB)	104,500 (595.5 KB)	15,815 (150.3 KB)
Minimum document size in words (KB)	26 (0.2 KB)	1 (0.002 KB)
Average document size in words (KB)	5,415 (32.9 KB)	91 (0.5 KB)

Table 1. Some key characteristics of the two corpora used for the evaluation

Neural Networks; we will refer to them as the *NN corpus*. To assess the generality of our approach, we have also conducted experiments with a larger and more varied corpus. The dbpedia (`http://dbpedia.org`) corpus –we will refer to it as *DBP corpus*– consists of more than 3 million documents that are extended abstracts from the Wikipedia website. The DBP corpus was chosen due to its differences to the NN corpus (smaller average document size, larger diversity in topics, wider vocabulary) and is used to demonstrate the performance of our protocols under a different setting. Table 1 summarises some key characteristics of the two corpora used in the evaluation.

All the experiments shown in this section were carried out using both document corpora. However, due to space considerations we report graphs for both corpora only when there exists a notable difference between the two experiments.

Query Sets. Since no database of continuous queries was available to us, we used two different methodologies to create continuous queries under model $\mathcal{AWPS}$.

The queries for the NN corpus are synthetically generated and consist of two parts: (i) a Boolean part containing atomic Boolean queries of the form $A \sqsupseteq wp$ and (ii) a VSM part containing atomic queries of the form $A \sim_k s$, where s is a text value. We set A to be TITLE, AUTHORS, ABSTRACT, or BODY with some probability. Subsequently, each atomic Boolean query of the form $A \sqsupseteq wp$ is generated using *words* and *technical terms* extracted automatically from the NN corpus using the C-value/NC-value approach of [66]. For more details of the methodology the interested reader can refer to [57, 52, 10]. An example of a user query created synthetically from the methodology briefly sketched above is:

$$\begin{gathered}(\text{AUTHOR} \sqsupseteq \mathit{Darwen}) \quad \wedge \\ (\text{TITLE} \sqsupseteq \mathit{implementation} \wedge (\mathit{RBF} \prec_{[0,3]} \mathit{networks})) \quad \wedge \\ \text{ABSTRACT} \sim_{0.6} \text{“Most work on the evolutionary approach to the iterated ...”}\end{gathered}$$

Since there is no publicly available database of continuous queries for the DBP corpus, we used 20.2 million Wikipedia article titles and categories (modified appropriately to fit our query language) as user queries. Each title or category represents one continuous query q that contains either a Boolean or a VSM part. For the case of the DBP corpus, we avoided synthetic creation of more complex queries (as done before for the case of the NN corpus) in order to demonstrate the performance of our methods under a different query setting.

Setup. We have implemented and experimented with eight variations of the DHTrie protocols: the iterative method It, the recursive method Re, the hybrid method $fixHy$, and the continuous splitting method Spl, which do not employ an FCache, and their counterparts that utilise an FCache (ItC, ReC, $fixHyC$, and $SplC$ respectively). The experiments with both corpora were conducted using the same machinery to enable the comparison across the different settings. All the methods and the DHTrie simulator were implemented in C/C++.

Parameter	Description	Baseline value
N	# of nodes in the system	10K-100K
Q	# of queries assigned to nodes	5M
C_s	# of entries in FCache	30K
C_t	# of publications used to train FCache	10K
W	average # of words per published document	5415 (NN), 91 (DBP)
SF	split factor (used for load balancing)	1, 10, 20, 30
T	split threshold (used for load balancing)	10
σ	size of recipients list ($fixHy$ method)	50 (NN), 10 (DBP))
π	percentage of recipients list ($perHy$ method)	4%

Table 2. Parameters varied in experiments, their descriptions, and their baseline values

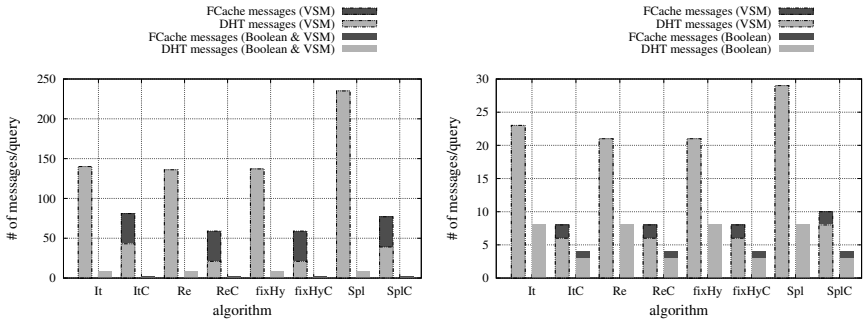
To carry out each experiment described in this section, we execute the following steps. Initially the network is set up by assigning keys to nodes. These keys are calculated using the SHA-1 cryptographic hash function and randomly created IP addresses and ports. After the network set up, we create 5M user queries and distribute them among the nodes using the protocol described in Section 3.1. According to the publication protocol, the number of posted queries does not affect the cost for publishing a document in the network; it only affects the matching time for the local filtering algorithms and the number of matching notifications produced (i.e., the higher the number of posted queries is, the higher the number of matching notifications produced). Table 2 summarises the parameters and the baseline values used for the experiments.

Evaluation Metrics. We are mainly interested in the performance of the eight different protocols in terms of *network traffic* and *latency* to publish a document or subscribe a query. To measure network traffic, we publish the corpus documents at different nodes and record the network activity. In our network, we can distinguish between two types of messages: messages sent through the DHT infrastructure and messages sent to a node using directly its IP address (FCache messages). In our experiments, we record and present the effects of both types of messages. Latency is measured in number of hops as follows. For each message (either publication or subscription) initiated by node P, we record the longest chain of messages needed until the message reaches all the intended recipients.

4.2 Varying the Type of Queries

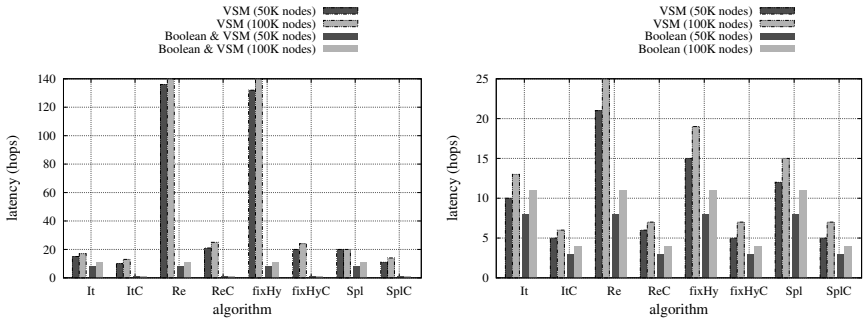
The first set of experiments investigates the cost of indexing a query in the network. For this setup we used two types of queries: (i) queries including only vector space atomic parts and (ii) queries with both Boolean and vector space parts. Indexing the second type of queries is the same as indexing queries with Boolean atomic parts only (see Section 3.1).

Each bar in Figure 3 shows the average message traffic recorded when indexing 500K queries of each type in a network of 50K nodes for both corpora. The most important observation in these graphs is that, regardless of the protocol and corpus applied on, vector space queries are more expensive to index than Boolean or mixed type queries. This happens because vector space queries are indexed at all nodes responsible for the distinct words in the query, contrary to other query types that are indexed under only one node (see Section 3.1). Notice the important role of FCache, the use of which manages the forwarding to the intended recipients of more than 1/3 of the total network traffic, thus relieving the DHT infrastructure of substantial messaging effort. It is clear that ReC and $fixHyC$ are the best performing protocols for vector space query indexing in



(a) NN corpus (b) DBP corpus

Fig. 3. Message traffic for indexing a query in the network

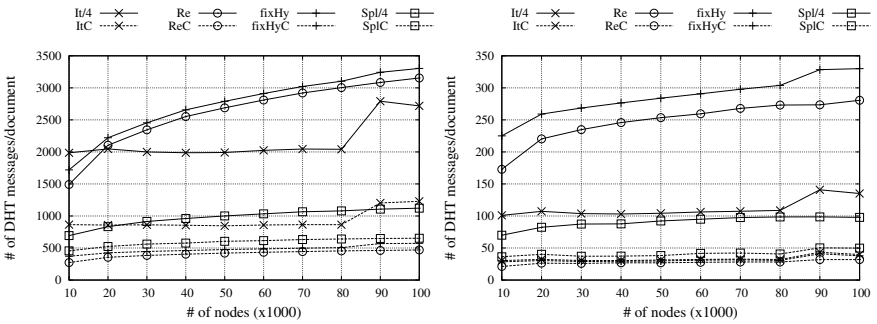


(a) NN corpus (b) DBP corpus

Fig. 4. Latency for indexing a query in the network

terms of message traffic. The difference in the message traffic induced by the queries of the two corpora is attributed to the different query lengths.

Figure 4 presents the publication latency achieved by all our protocols when indexing 500K (vector space or mixed) queries in a network of 50K or 100K nodes for both corpora. As we can see in this figure, latency in the indexing of Boolean or mixed type queries is invariant since they are indexed under only one node. For the vector space queries however, one important observation is the low latency of the iterative and the continuous splitting methods, and the high latency of the recursive ones. This is due to the routing infrastructure used and the specifics of each method. The iterative methods use a single lookup message for each one of the intended recipients of the query, thus parallelising the subscription process. The continuous splitting methods split the recipient lists and adapt the subscription process to the finger tables of the forwarding nodes. On the other hand, the recursive method uses long recipients lists and contacts them in a recursive way, thus increasing subscription latency. Additionally, protocol $fixHy$ seems to behave similarly to Re in terms of latency, which is explained by the fact that in this experiment the two protocols have roughly the same size of recipients lists. This happens because Re creates small recipients lists (due to the size of the query) and thus, the size of the list is similar to the size we



(a) NN corpus (b) DBP corpus

Fig. 5. Message traffic for various network sizes

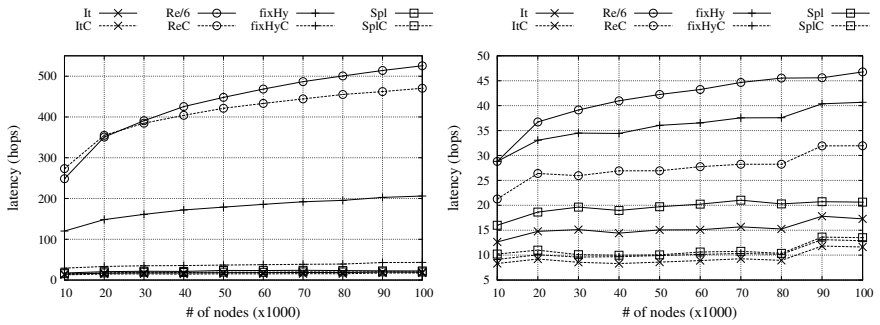
use for protocol $fixHy$. FCache, similarly to message traffic, plays an important role by reducing latency (up to 60%) for all the protocols.

4.3 Varying Network Size

Although query indexing performance is important, in an IF scenario resource publication is the time critical component. This second set of experiments targets the performance of the protocols in terms of message traffic and publication latency for different network sizes. In this experiment, we randomly selected 100 documents from the NN and the DBP corpus and used them as publications by randomly assigning each one to a different publisher node for each of the 10 different runs used for averaging measurements. Having published the documents, we recorded the total number of DHTrie messages generated by the network in order to match these documents against the indexed user queries.

In Figure 5, the performance of the protocols in terms of DHTrie messages/document for both corpora is shown. The main observation is that the number of messages generated by all protocols grows at a logarithmic scale mainly due to the routing infrastructure used. A second observation emerging from the graph is the effectiveness of the FCache independently of the message routing protocol used and the corpus it is applied at. The use of FCache results in the reduction of messages sent using the routing infrastructure by more than 6 times for NN corpus (resp. 7 times for the DBP corpus) in the recursive, the hybrid and the continuous splitting method, and by 8 times for the NN corpus (resp. 4 times for the DBP) in the iterative method. Notice that the improvement in the performance of the protocols when using the FCache is slightly lower for the DBP corpus (compared to the NN corpus) due to the significantly smaller document size and the wider vocabulary (because of lower FCache utilisation and thus higher DHT traffic). Finally, notice that the number of DHT messages needed to index a document from the DBP corpus is significantly lower than that of the NN corpus, due to the significantly smaller average document size.

In Figure 6, the performance of the different protocols in terms of publication latency for both corpora is shown. Similarly to the previous set of experiments, low latency is observed when using the iterative or the continuous splitting methods, whereas high latency is caused by the recursive ones. The use of the FCache



(a) NN corpus (b) DBP corpus

Fig. 6. Latency for various network sizes

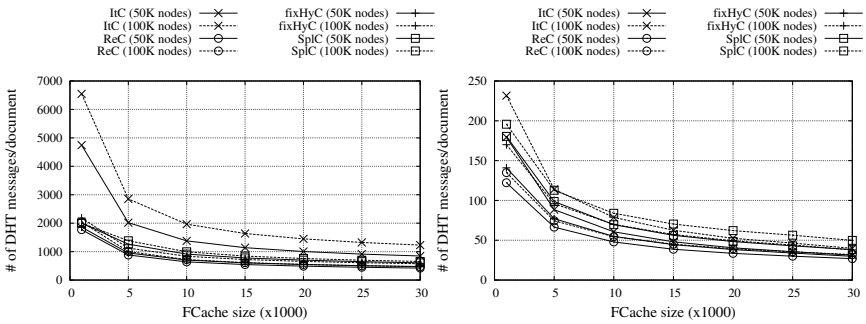
reduces publication latency for both corpora by shortening the intended recipients lists of ReC and $fixHyC$. Additionally, it is worth pointing out that the smaller document size of the DBP corpus results in lower publication latency compared to that of the NN corpus, due to the smaller recipients lists.

Finally, in our measurements (graph not shown due to space reasons) the lowest processing cost per document for a network size of 100K nodes for the NN corpus (resp. DBP corpus) is presented for method ReC with about 1,300 (resp. 42) messages in total, with about 65% of them being FCache messages, as opposed to 40% for method $SplC$, and 55% for method ItC for both corpora.

4.4 Varying the FCache Size

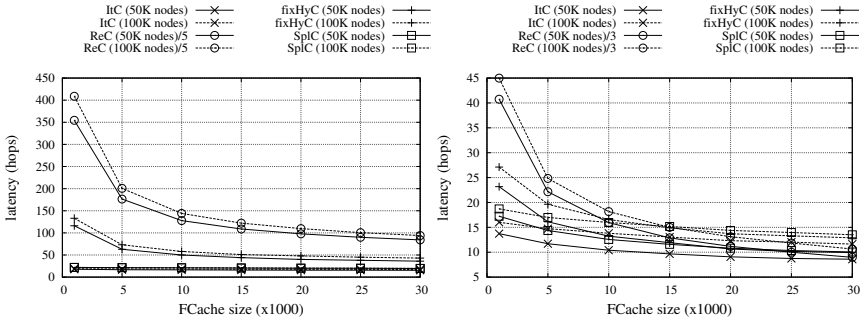
The third set of experiments targeted the performance of the protocols under different FCache sizes, and studied the effect of FCache in message traffic and publication latency. Initially, we used (a part of) the document corpora as training sets for populating the FCache of the different nodes; a randomly chosen node P publishes 10K documents and populates its FCache with the IP addresses of the nodes that are responsible for the most frequent words contained in the published documents. Then, another 100 documents are published by P and the size of the FCache is limited to different values. Subsequently, the total number of messages used to match these documents against the stored user queries is recorded and averaged over 10 runs with different nodes. Figure 7 shows the messages traffic per document for the two corpora as the size of the FCache grows.

As shown in Figure 7, the number of messages sent using the DHTrie routing infrastructure reduces quickly for both corpora as the size of FCache increases, and the decrease rate depends on FCache size due to the skewness in the corpus vocabulary. This results in reaching an FCache size after which no significant effect is observed in message traffic reduction (around 30K entries, the rightmost point on the x-axis). Additionally, the reduction factor for all methods and for both corpora is similar: 40-50% (resp. 10-15%) reduction in message costs depending on the method for small (resp. large) FCache sizes. Notice also that for protocols ReC, HyC, and $SplC$ the performance of FCache remains almost constant for different network sizes, whereas for protocol ItC 50% more DHTrie messages/document are needed for an 100% increase in network size. Finally,



(a) NN corpus (b) DBP corpus

Fig. 7. Message traffic for different FCache sizes

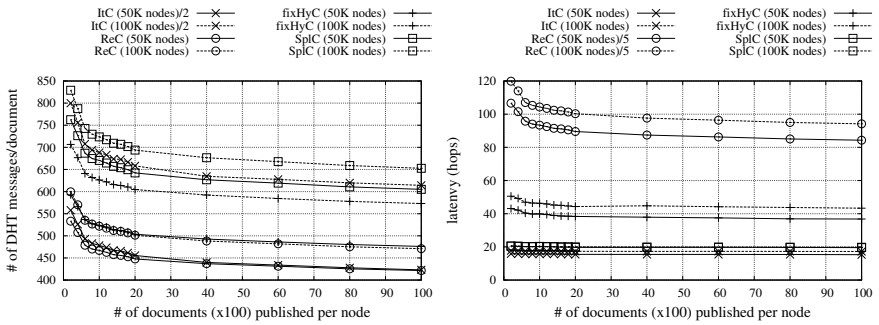


(a) NN corpus (b) DBP corpus

Fig. 8. Latency for different FCache sizes

notice that the number of DHT messages needed to index a document from the DBP corpus is significantly lower than that of the NN corpus due to the significant difference in average document size.

In Figure 8, we show the publication latency for the different protocols and the way it is affected by the variation of the FCache size. As expected, the reduction in the latency for all the protocols is lower as the FCache size increases due to the skewness of the vocabulary entries used for populating the FCache. Additionally, not all protocols are affected in the same way from FCache increase in size. *ItC* and *SplC* remain relatively unaffected for both corpora by the increase both in FCache size and in network size, something that is also verified from the graphs of the previous section. This is due to the routing infrastructure and the parallel way of publishing the incoming documents. Contrary, protocols *ReC* and $fixHyC$ seem to perform better when the size of the FCache increases, since this causes reduction in the size of recipients lists. Moreover, the reduction factor across corpora is similar: low for methods *ItC* and *SplC*, while it reaches 35-45% (resp. 5-15%) for small (resp. large) FCache sizes for methods *ReC* and $fixHyC$. Finally, FCache is equally utilised by all methods (graph omitted due to space reasons), as the number of FCache messages/document is similar for all methods and reaches up to 900 (resp. 19) FCache messages/document for the NN corpus (resp. DBP corpus) for 30K entries.



(a) Message traffic (b) Latency

Fig. 9. Performance for different levels of FCache training

4.5 Effect of FCache Training

In this set of experiments, we measure the effect of FCache training on message cost and publication latency in the NN corpus (results for the DBP corpus are similar and are omitted due to space reasons). To do so, we randomly selected a node P and trained its FCache with a varying number of documents. In this way, the node was able to collect statistics about frequent words used in document publications and populate its FCache with pointers to frequently contacted nodes. Subsequently, we published 100 documents to P and recorded the average message cost and publication latency. The results shown in Figure 9 are averaged over 100 runs for different nodes to eliminate network topology effects.

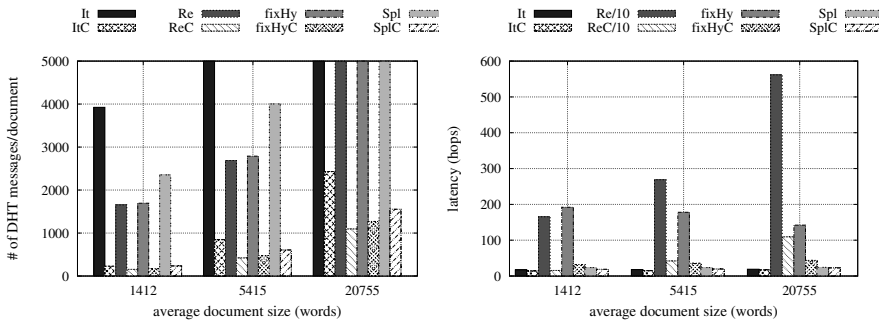
Figure 9(a) shows that the performance of all protocols improves as more documents get published. Methods *ReC* and $fixHyC$ are less sensitive in this parameter, as the difference in the number of messages observed is about 100 messages for 50 times more documents (the leftmost and rightmost point in the x-axis). Additionally, *ReC* and *SplC* show less sensitivity with respect to the network size, contrary to ItC that needs about 50% more messages. Finally, all methods show a similar behaviour for the two network sizes tested.

Figure 9(b) shows the effect of the number of publications in latency. We observe that method *ReC* is the most affected by the training level of the FCache, as it is heavily dependent on the FCache information to reduce long recipients lists. Method $fixHyC$ is less affected as it produces shorter recipient lists than *ReC*, while methods *SplC* and ItC remain unaffected due to the protocol design. Additionally, all methods present a slight increase in message traffic when doubling the network size due to the logarithmic routing.

Finally, the number of FCache hits for the NN corpus (resp. DBP corpus) and for all methods is between 830 and 875 (resp. 12 and 20) messages/document for a network of 50K nodes. This shows that FCache hits are only affected by the size and skewness of the published data, not by the protocol used.

4.6 Varying the Document Size

Document (i.e., publication) size is an important parameter in the performance of our protocols. This set of experiments targeted the performance of the protocols under various document sizes. Due to space reasons we show only the



(a) Message traffic (b) Latency

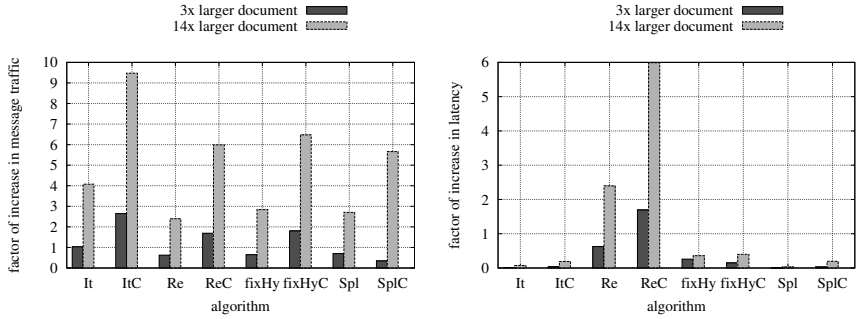
Fig. 10. Performance for documents of different size

experiments on the NN corpus that has a larger variation in document size. The findings for the DBP corpus are briefly summarised in the text and are in line with the ones presented here. Figure 10 shows the message cost and latency for publishing documents of varying size for all protocols. Each bar is an average of the message cost and latency (appropriately truncated to show the best performing methods) for 100 documents, published by 1,000 different nodes (in a network of 50K nodes in total) to normalise network topology effects.

Figure 10(a) shows that for small documents, methods *Re*, $fixHy$, and *Spl* achieve 50% less message traffic than It, while all FCache variations of the protocols perform similarly. This happens because in smaller documents there will be less infrequent words that may result in FCache misses. However, as document size increases the importance of the message forwarding method is more obvious (i.e., notice that ReC is able to process documents of 21K words by using only 1,000 messages). Note also that although protocols ReC, $fixHyC$, and $SplC$ perform similarly in terms of message traffic, as discussed later in this section, $SplC$ handles latency better than its counterparts. Our findings for the DBP corpus are similar, but, since the average document size is significantly smaller, message traffic is about 10 times lower (see also Figure 5).

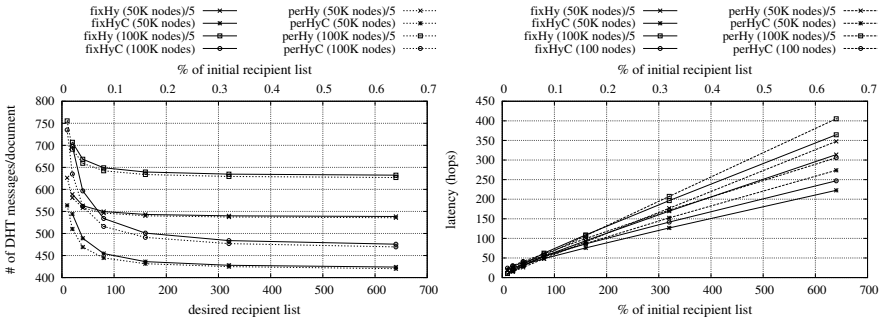
Figure 10(b) shows how document size affects latency for the different protocols. The most important observation is the inefficient performance of the *Re* and ReC protocols (notice that measurements are reduced by a factor of 10 for readability), which shows the dependence of both methods on document size (that increases the size of recipients lists). Contrary, the rest of the protocols are insensitive to document size, for different reasons each: It and ItC because of the lack of recipient lists, $fixHy$ and $fixHyC$ because of document size-independent recipient lists, and *Spc* and $SplC$ because of the adaptivity of the forwarding process. The measurements for the DBP corpus showed a similar behaviour for all methods due to the small average document size and thus recipient list size.

We also examined the relative increase in message traffic and latency for three groups of documents, D_1, D_2, and D_3, where D_2 is 3 times larger and D_3 is 14 times larger on average than D_1. Initially, 100 random nodes were chosen to publish documents from group D_1, and the message traffic and latency were recorded. Then, the other two document groups were published in the same way, and the measurements were recorded and compared to those of group D_1. Figure 11 shows the factor of increase in message traffic and latency for each



(a) Message traffic (b) Latency

Fig. 11. Increase rate for different document sizes



(a) Message traffic (b) Latency

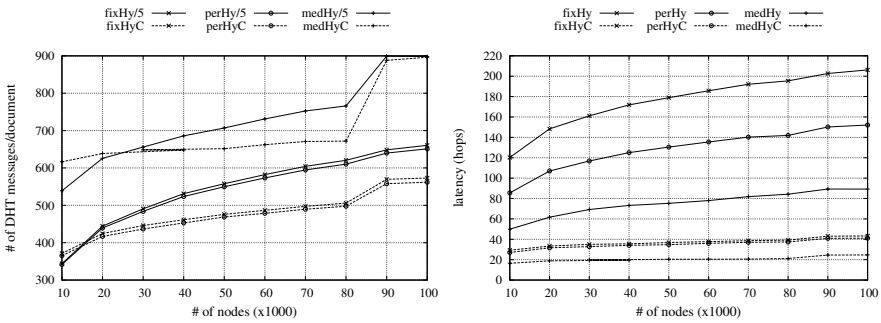
Fig. 12. Performance when varying σ ($fixHy$) and π ($perHy$)

protocol when publishing the two different groups of documents. Our findings on the sensitivity of the methods to document size are aligned with those of Figure 10: in terms of message traffic ReC, $fixHyC$, and $SplC$ show low sensitivity to document size, while in terms of latency Re and ReC are highly sensitive.

4.7 Comparison of the Hybrid Methods

This set of experiments aims at comparing message overhead and publication latency of the hybrid method variants by examining the correlations between the parameters σ and π of methods $fixHy$ and $perHy$, and the performance of the parameterless method $medHy$. Figure 12 demonstrates the performance of the $fixHy$ and $perHy$ methods for two different network sizes (50K and 100K nodes). Each point is averaged over 10 runs, and 100 NN corpus documents, randomly assigned to publisher nodes, were used as incoming publications. The findings for the DBP corpus are similar and are omitted for space reasons.

Figure 12(a) shows the average number of DHT messages needed to publish a large document as the desired recipient list size increases for the hybrid methods. To interpret the results of this graph the reader is reminded that the hybrid protocols try to combine the iterative and recursive protocols: the shorter the recipients list size is, the closer the protocol is to the iterative counterpart (notice network traffic reduction as the recipient list size increases). Additionally, FCache reduces network traffic and the effect of network size significantly,



(a) Message traffic (b) Latency

Fig. 13. Performance of the hybrid methods for various network sizes

while making the methods less sensitive to parameter changes. Finally, notice that although the splitting of the recipients list is performed in a different way by $fixHy$ and $perHy$, parameters σ and π have a similar effect since they relate through the average publication size (i.e., it is possible to adjust π so that in the average case it will split the message in pieces of average size σ). Notice however, that π is easier to set than σ as it does not require any knowledge on the specifics of the published documents.

Publication latency is linear to the increase in the recipient list size (Figure 12(b)) for both methods, while FCache manages to keep latency low.

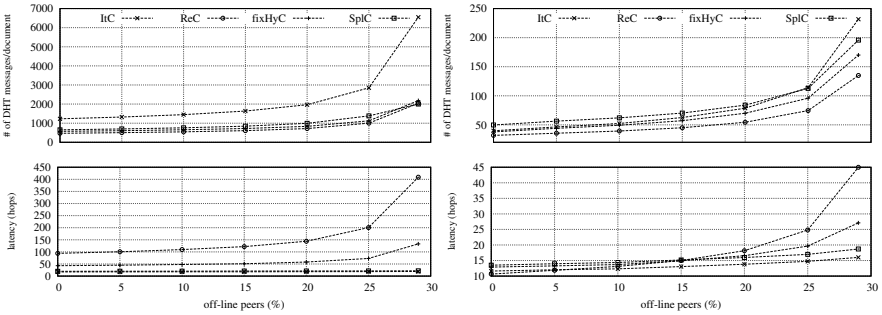
Finally in Figures 13(a) and (b), we demonstrate message traffic and latency for all hybrid variations. Message traffic for all methods grows logarithmically due to the routing infrastructure, while the introduction of FCache results in a significant decrement in message traffic. As the $medHy$ method is by design aimed towards optimising latency (due to the way of splitting recipients lists), $fixHyC$ and $perHyC$ have also been set with latency in mind ($\sigma = 50$, $\pi = 0.04$) for comparison reasons. Finally, notice that the $fixHy$ and $perHy$ methods perform similarly in terms of message traffic, but differ in latency, which demonstrates the importance of optimising this tradeoff and constituted the main driver for the introduction of *Spl* and *SplC* methods.

4.8 Effect of Node Churn

In this section, we target the performance of the protocols in terms of message traffic and publication latency under node churn by introducing a short life span for a varying percentage of nodes in the network.

In Figure 14, our measurements show that when 5% of the nodes are off-line during a lookup, the message cost increase is no more than 8% for the NN corpus (12% for the DBP corpus), showing that FCache is able to cope up with misses. On the other hand, when 30% of the nodes are off-line, the message cost increases significantly for both corpora, since for each FCache miss several DHT messages have to be issued. Remember though, that FCache misses affect only network traffic and not the correctness of the protocols. Additionally, all methods, apart from *ReC* (that packs messages together and increases the recipients list size), present a good overall performance in terms of latency.

To deal with node failures for methods that do not rely on FCache (i.e., *It*, *Re*, $fixHy$, and *Spl*) we reside on DHT mechanisms, which can guarantee



(a) NN corpus (b) DBP corpus

Fig. 14. Message traffic and latency for 100K nodes under churn

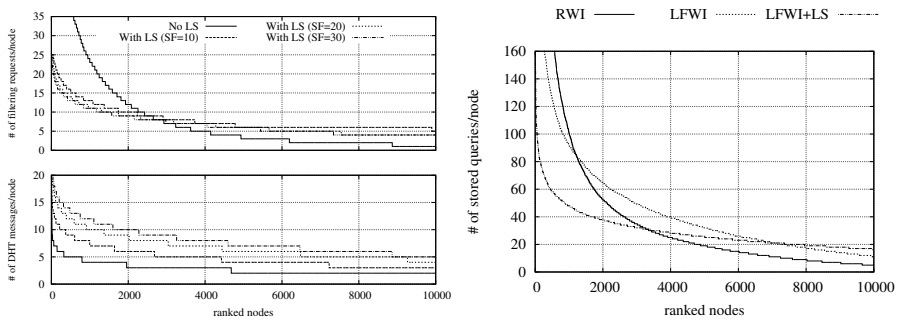
correctness of lookups and finger table repairs. All measurements for node churn in Chord [56] carry over to our setting; for more details the reader is referred to [56] due to space reasons.

4.9 Skewed Data Distributions and Load Balancing

In typical IR scenarios the word distributions associated with documents and queries are typically skewed. In a pub/sub setting, *load balancing* becomes a key issue when trying to partition the query space among the different nodes of a DHT. We can distinguish three types of node load: *query load* (i.e., the number of queries stored at a node), *routing load* (i.e., the number of messages a node forwards due to the protocols), and *filtering load* (i.e., the number of publications a node has to filter against the stored queries).

Balancing the Filtering Load. In the DHT literature, work on load balancing has concentrated on two particular problems: (i) *address-space load balancing* concerning how to partition the address-space of a DHT "evenly" among keys; it is typically solved by relying on consistent hashing and constructions such as virtual servers [8] or potential nodes [67] and (ii) *item load balancing* addressing how to balance load in the presence of data items with arbitrary load distributions [67, 68] as in our case.

We have implemented and evaluated a simple algorithm based on the well-known concept of *load-shedding* (LS), where an overloaded node attempts to offload work to less loaded nodes. Once a node P understands that it has become overloaded, it chooses the most frequent word w it is responsible for and a small integer k. Then P contacts the nodes responsible for words wj for all j, $1 \leq j \leq k$, where wj is the concatenation of strings w and j, and asks them to be its replicas. Then P notifies the rest of the network about this change in responsibilities by piggy-backing the necessary information in DHTrie maintenance messages. Each node M that receives this message notes down the word w. Later on, if M has a new publication containing w, it divides the filtering responsibility for w among P and k other nodes by concatenating a random number from 1 to k to the end of w and using DHTrie to find the node responsible for the concatenated word. In this way, the filtering responsibility of w for P is reduced by $k+1$ times (node P and k new nodes). We call $k+1$ the *split factor* (SF) in subsequent experiments.



(a) Filtering and routing load (b) Query load

Fig. 15. Load distribution for the 10K most loaded nodes

Figure 15(a) shows the average number of filtering requests (top part) received by each node in a time window T for a period of 100*T. SF was varied between 10 and 30 nodes and T was set to 10 filtering requests. We also varied T but did not observe significant differences in the load distribution.

Balancing the Routing Load. Balancing the filtering load causes an increase in message traffic due to FCache misses and thus, the overall routing load of the system is increased. Figure 15(a) shows the number of routing requests (bottom part) received by the 10K most loaded nodes in the network. The number of DHT messages/document increases after the load balancing algorithm is run (80% for SF=10, 180% for SF=20, and 240% for SF=30), but the new load imposed on the network is well distributed among the nodes and does not cause overloading in any specific group of nodes.

Balancing the Query Load. Even query distribution among nodes is a hard task to achieve since typically queries follow a skewed word distribution. To distribute the queries to the nodes responsible, we utilised two different query indexing methods. The first method, coined RWI (Random Word Index), follows the subscription protocol of Section 3.1, where a node P indexes a query q to the node responsible for a *randomly* selected word w contained in any of the text values $s_1, \ldots, s_m$ or word patterns $wp_{m+1}, \ldots, wp_n$ of q. Contrary, the second method, coined $LFWI$ (Least Frequent Word Index), takes into account the *document frequency* of the words contained in q and indexes q to the node responsible for the *least frequent* word w contained in it.

Notice that the methods described above are orthogonal to the routing method utilised by P to forward the query, since they are only used to select under which word q will be indexed in the network. The intuition behind method $LFWI$ is to index the query under the node responsible for the least frequent word in it, thus avoiding the overload of nodes responsible for popular terms. Figure 15(b) shows the results for the 10K most loaded nodes and 1M queries indexed in a network of 50K nodes and each graph is produced as an average over 10 runs. Method *LFWI+LS*, i.e., the combination of the $LFWI$ and LS methods (for with SF=10), achieves the most uniform load distribution of all approaches.

4.10 Summing Up

In all experiments, the methods with the FCache (*ItC*, *ReC*, $fixHyC$, *SplC*) outperformed (both in message traffic/latency) their counterparts without the FCache (*It*, *Re*, $fixHy$, *Spl*) showing the usefulness of the proxying mechanism.

When message traffic is the optimisation metric (at the expense of latency), method *ReC* is the best candidate, as it is less sensitive to network and publication size. Contrary, when latency is the optimisation metric (at the expense of message traffic), method *ItC* presents the best alternative, as it is less affected by network size, FCache size/training, and publication size. Hybrid methods $fixHyC$ and $perHyC$ are tunable alternatives to the *ItC* and *ReC* methods, adjustable to publication size, and offer a good tradeoff between message traffic and latency, while method $medHyC$ is the parameterless variation of the hybrid family that slightly favors latency over message traffic. Finally, method *SplC* is slightly more expensive than *ReC* in network traffic, but its latency is as low as the best performing method *ItC*. Moreover, *SplC* is less sensitive than the hybrid methods to changes in network size and FCache size/training.

Overall, $perHyC$ and *SplC* are the two most versatile and well-performing protocols that put emphasis both on optimising message traffic and latency. Method $perHyC$ may be adjusted in a per-node fashion, however parameter setting may require background knowledge of publication characteristics. On the other hand, *SplC* is an adaptable and versatile method that performs well under many different scenarios (including node churn) and can be deployed off-the-shelf, without any need for parameter setting.

5 Conclusions and Outlook

In this work, we have presented and evaluated a set of protocols that efficiently extend Chord with pub/sub functionality and introduced proxying and load balancing mechanisms to cope with message traffic, latency, and skewness of data. The results of the earlier version of this paper [10] have influenced most of our work on P2P computing over the last years, inspiring us to develop IF functionality [27] in the Minerva system [53], study IF in an XML context [69], design DHT-based digital libraries [52], and implement Web/Grid service registries [70] for the EU projects OntoGrid and SemsorGrid4Env. The deployment of these ideas on various domains demonstrates the generality of the problem and shows that our protocols may be applied beyond the adopted scenario.

Lately, MapReduce [71] is widely used as the programming paradigm to achieve distributed data analysis, load balancing, and fault tolerance by parallelising map and reduce operations in the cloud. We plan to port our work to the MapReduce paradigm (e.g., following the philosophy of Memcached for a generic distributed service) to allow the deployment of our protocols in a well-known computing paradigm aiming for higher penetration in domains other than P2P. Additionally, such an implementation will encourage the usage and evaluation of the protocols in real-life scenarios and allow us to get usage data involving performance measurements, real user profiles, and publishing behaviour patterns.

Furthermore, the deployment of our protocols in large-scale distributed social networks would allow novel data management functionality, like subscriptions

over content/tags with aggregation (e.g., notify me when a published document matches my continuous query and k of my friends have tagged it as interesting).

References

1. Hameurlain, A., Hussain, F., Morvan, F., Tjoa, A.M., eds.: Data Management in Cloud, Grid and P2P Systems, Globe, Springer (2012)
2. Sinha, V., Gupta, A., Kohli, G.: Comparative Study of P2P and Cloud Computing Paradigm Usage in Research Purposes. In: CNC. (2011)
3. Kavalionak, H., Montresor, A.: P2P and Cloud: A Marriage of Convenience for Replica Management. In: IWSOS. (2012)
4. Trajkovska, I., Salvachua Rodriguez, J., Mozo Velasco, A.: A novel P2P and cloud computing hybrid architecture for multimedia streaming with QoS cost functions. In: ACM Multimedia. (2010)
5. Kontominas, D., Raftopoulou, P., Tryfonopoulos, C., Petrakis, E.G.: DS4: A Distributed Social and Semantic Search System. In: ECIR. (2013)
6. Loupasakis, A., Ntarmos, N., Triantafillou, P.: eXO: Decentralized Autonomous Scalable Social Networking. In: CIDR. (2011)
7. Graffi, K., Gross, C., Mukherjee, P., Kovacevic, A., Steinmetz, R.: LifeSocial.KOM: A P2P-Based Platform for Secure Online Social Networks. In: P2P. (2010)
8. Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: ACM SIGCOMM. (2001)
9. Koubarakis, M., Skiadopoulos, S., Tryfonopoulos, C.: Logic and Computational Complexity for Boolean Information Retrieval. IEEE TKDE (2006)
10. Tryfonopoulos, C., Idreos, S., Koubarakis, M.: Publish/Subscribe Functionality in IR Environments using Structured Overlay Networks. In: ACM SIGIR. (2005)
11. Carzaniga, A., Rosenblum, D.S., Wolf, A.: Design and Evaluation of a Wide-Area Event Notification Service. ACM TOCS (2001)
12. Koubarakis, M., Tryfonopoulos, C., Idreos, S., Drougas, Y.: Selective Information Dissemination in P2P Networks: Problems and Solutions. SIGMOD Record (2003)
13. Rowstron, A., Kermarrec, A.M., Castro, M., Druschel, P.: Scribe: The Design of a Large-scale Event Notification Infrastructure. In Crowcroft, J., Hofmann, M., eds.: COST264. (2001)
14. Pietzuch, P., Bacon, J.: Hermes: A Distributed Event-Based Middleware Architecture. In: DEBS. (2002)
15. Tam, D., Azimi, R., Jacobsen, H.A.: Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables. In: DBISP2P. (2003)
16. Terpstra, W., Behnel, S., Fiege, L., Zeidler, A., Buchmann, A.: A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In: DEBS. (2003)
17. Gedik, B., Liu, L.: PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In: ICDCS. (2003)
18. Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., Panigrahy, R.: Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In: ACM STOC. (1997)
19. Bender, M., Michel, S., Parkitny, S., Weikum, G.: A Comparative Study of Pub/Sub Methods in Structured P2P Networks. In: DBISP2P. (2006)
20. Triantafillou, P., Aekaterinidis, I.: Content-based publish-subscribe over structured P2P networks. In: DEBS. (2004)
21. Gupta, A., Sahin, O.D., Agrawal, D., Abbadi., A.E.: Meghdoot: Content-Based Publish/Subscribe over P2P Networks. In: Middleware. (2004)
22. Aekaterinidis, I., Triantafillou, P.: PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network. In: ICDCS. (2006)

23. Aekaterinidis, I., Triantafillou, P.: Internet scale string attribute publish/subscribe data networks. In: CIKM. (2005)
24. Tran, D., Pham, C.: Enabling content-based publish/subscribe services in cooperative P2P networks. Computer Networks (2010)
25. Lo, S.C., Chiu, Y.T.: Design of content-based publish/subscribe systems over structured overlay networks. IEICE Transactions (2008)
26. Liau, C., Ng, W., Shu, Y., Tan, K.L., Bressan, S.: Efficient Range Queries and Fast Lookup Services for Scalable P2P Networks. In: Databases, Information Systems, and Peer-to-Peer Computing. LNCS. (2005)
27. Tryfonopoulos, C., Zimmer, C., Koubarakis, M., Weikum, G.: Architectural Alternatives for Information Filtering in Structured Overlay Networks. IEEE Internet Computing (2007)
28. Zheng, X., Luo, J., Cao, J.: Pat: A P2P Based Publish/Subscribe System for QoS Information Dissemination of Web Services. In: ICWS. (2009)
29. Cheung, A.Y., Jacobsen, H.A.: Load balancing content-based publish/subscribe systems. ACM TOCS (2010)
30. Bernard, S., Potop-Butucaru, M., Tixeuil, S.: A framework for secure and private p2p publish/subscribe. In: SSS. (2010)
31. Drosou, M., Stefanidis, K., Pitoura, E.: Preference-aware publish/subscribe delivery with diversity. In: DEBS. (2009)
32. Tang, C., Xu, Z.: pFilter: Global Information Filtering and Dissemination Using Structured Overlays. In: FTDCS. (2003)
33. Zhu, Y., Hu, Y.: Ferry: A P2P-Based Architecture for Content-Based Publish/Subscribe Services. IEEE TPDS (2007)
34. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-addressable Network. In: ACM SIGCOMM. (2001)
35. Kossmann, D.: The State of the art in distributed query processing. ACM Computing Surveys (2000)
36. Stonebraker, M., Aoki, P., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., Yu, A.: Mariposa: A Wide-Area Distributed Database System. VLDB Journal (1996)
37. Litwin, W., Neimat, M.A., Schneider, D.A.: LH* - A Scalable, Distributed Data Structure. ACM TODS (1996)
38. Balakrishnan, H., Kaashoek, M., Karger, D., Morris, R., Stoica, I.: Looking up data in P2P systems. CACM (2003)
39. Huebsch, R., Hellerstein, J., Lanham, N., Loo, B., Shenker, S., Stoica, I.: Querying the Internet with PIER. In: VLDB. (2003)
40. Harren, M., Hellerstein, J., Huebsch, R., Loo, T., Shenker, S., Stoica, I.: Complex Queries in DHT-based Peer-to-Peer Networks. In: IPTPS. (2002)
41. Idreos, S., Tryfonopoulos, C., Koubarakis, M.: Distributed Evaluation of Continuous Equi-join Queries over Large Structured Overlay Networks. In: ICDE. (2006)
42. Palma, W., Akbarinia, R., Pacitti, E., Valduriez, P.: DHTJoin: processing continuous join queries using DHT networks. DPD (2009)
43. Dedzoe, W., Lamarre, P., Akbarinia, R., Valduriez, P.: Efficient Early Top-k Query Processing in Overloaded P2P Systems. In: DEXA. (2011)
44. Cai, M., Frank, M., Yan, B., MacGregor, R.: A subscribable peer-to-peer RDF repository for distributed metadata management. Journal of Web Semantics (2004)
45. Liarou, E., Idreos, S., Koubarakis, M.: Continuous RDF Query Processing over DHTs. In: ISWC. (2007)
46. Lohrmann, B., Battré, D., Kao, O.: Towards Parallel Processing of RDF Queries in DHTs. In: Globe. (2009)
47. Battré, D., Heine, F., Höing, A., Hovestadt, M., Kao, O., Liebetruth, C.: Dynamic Knowledge in DHT Based RDF Stores. In: SWWS. (2008)

48. Belkin, N., Croft, W.: Information Filtering and Information Retrieval: Two Sides of the Same Coin? CACM (1992)
49. Li, J., Loo, B., Hellerstein, J., Kaashoek, M., Karger, D., Morris, R.: On the Feasibility of Peer-to-Peer Web Indexing and Search. In: IPTPS. (2003)
50. Reynolds, P., Vahdat, A.: Efficient Peer-to-Peer Keyword Searching. In: Middleware. (2003)
51. Hsiao, H.C., King, C.T.: Similarity Discovery in Structured P2P Overlays. In: ICPP. (2003)
52. Tryfonopoulos, C., Idreos, S., Koubarakis, M.: LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs. In: ECDL. (2005)
53. Bender, M., Michel, S., Triantafillou, P., Weikum, G., Zimmer, C.: MINERVA: Collaborative P2P Search (Demo). In: VLDB. (2005)
54. Gounaris, A., Fernandes, A., Papadopoulos, A., C.Yfoulis: Parallel Query Processing on the Grid. In: Advances in Parallel Computing. (2009)
55. Narendula, R., Papaioannou, T., Aberer, K.: My3: A highly-available P2P-based online social network. In: P2P. (2011)
56. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. IEEE/ACM TON (2003)
57. Tryfonopoulos, C., Koubarakis, M., Drougas, Y.: Information Filtering and Query Indexing for an Information Retrieval Model. ACM TOIS (2009)
58. Yan, T., Garcia-Molina, H.: The SIFT Information Dissemination System. ACM TODS (1999)
59. Huebsch, R.: Content-Based Multicast: Comparison of Implementation Options. Technical Report UCB//CSD-03-1229, UC Berkeley (2003)
60. Pitoura, T., Ntarmos, N., Triantafillou, P.: Replication, Load Balancing and Efficient Range Query Processing in DHTs. In: EDBT. (2006)
61. Gopalakrishnan, V., Silaghi, B., Bhattacharjee, B., Keleher, P.: Adaptive Replication in Peer-to-Peer Systems. ICDCS (2004)
62. Shen, H.: Efficient and Effective File Replication in Structured P2P File Sharing Systems. In: P2P. (2009)
63. Deb, S., Linga, P., Rastogi, R., Srinivasan, A.: Accelerating Lookups in P2P Systems using Peer Caching. In: ICDE. (2008)
64. Bhattacharjee, B., Chawathe, S., Gopalakrishnan, V., Keleher, P., Silaghi, B.: Efficient Peer-To-Peer Searches Using Result-Caching. In: IPTPS. (2003)
65. Dong, L.: Automatic term extraction and similarity assessment in a domain specific document corpus. Master's thesis, Department of Computer Science, Dalhousie University (2002)
66. Frantzi, K., Ananiadou, S., Mima, H.: Automatic recognition of multi-word terms: the C-value/NC-value method. IJDL (2000)
67. Karger, D.R., Ruhl, M.: Simple efficient load balancing algorithms for peer-to-peer systems. In: SPAA. (2004)
68. Datta, A., Schmidt, R., Aberer, K.: Query-load balancing in structured overlays. In: CCGRID. (2007)
69. Miliaraki, I., Kaoudi, Z., Koubarakis, M.: XML Data Dissemination using Automata on Top of Structured Overlay Networks. In: WWW. (2008)
70. Kaoudi, Z., Koubarakis, M., Kyzirakos, K., Miliaraki, I., Magiridou, M., Papadakis-Pesaresi, A.: Atlas: Storing, Updating and Querying RDF(S) Data on Top of DHTs. Journal of Web Semantics (2010)
71. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI. (2004)