Load-Balanced Query Dissemination in Privacy-Aware Online Communities

Emiran Curtmola, Alin Deutsch
UC San Diego
{ecurtmola,deutsch}@cs.ucsd.edu

K.K. Ramakrishnan, Divesh Srivastava
AT&T Labs – Research
{kkrama,divesh}@research.att.com

ABSTRACT

We propose a novel privacy-preserving distributed infrastructure in which data resides only with the publishers owning it. The infrastructure disseminates user queries to publishers, who answer them at their own discretion. The infrastructure enforces a *publisher k-anonymity* guarantee, which prevents leakage of information about which publishers are capable of answering a certain query. Given the virtual nature of the global data collection, we study the challenging problem of efficiently locating publishers in the community that contain data items matching a specified query. We propose a distributed index structure, UQDT, that is organized as a union of Query Dissemination Trees (QDTs), and realized on an overlay (i.e., logical) network infrastructure. Each QDT has data publishers as its leaf nodes, and overlay network nodes as its internal nodes; each internal node routes queries to publishers, based on a summary of the data advertised by publishers in its subtrees. We experimentally evaluate design tradeoffs, and demonstrate that UQDT can maximize throughput by preventing any overlay network node from becoming a bottleneck.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Distributed databases*

General Terms: Design, Performance

1. INTRODUCTION

During the last decade, the web has enabled unparalleled access to the vast amount of electronic data that is continually being created, and search engine technology has made it feasible to issue queries and locate web sites that contain data of interest to a user.

As the web evolves, two significant new trends are emerging. First, write access to the web is becoming increasingly democratic as it is easier for a large number of users to create and publish data on a wide variety of topics; this is evident from the proliferation of blogs, Wikis (e.g., Wikipedia), user-generated videos and photos, etc. Second, it is becoming easier to form web communities based on shared interests; this is evident in the considerably popularity of social networking sites like Facebook and MySpace. With the confluence of these two trends comes the natural desire to freely exchange data within the community – this includes making one's own data collection accessible to others within the community, and also being able to query, tag, and comment on the global collection that is the union of all local data of users within the community.

Recent events have called attention to the pressing need to enhance the infrastructure of online communities to enable freedom of speech without fear of retribution against the community users. People have come to learn that their online blogs along with the mainstream news websites can be easily censored, or worse, the true identity behind their online nicknames can be revealed. This information can be used to censor or discriminate certain individuals pertaining to various online activist groups or dissidents. To fully deliver on the promise of freely exchanging data, any community-supporting infrastructure needs to enforce the key requirement of preserving the privacy of publishers. That is, there should be no easy way for any third party to infer the identity of publishers of documents on specific topics.

This privacy-preserving publishing requirement precludes some obvious approaches that reuse and build on existing centralized technologies, e.g., search engines, hosted online communities, etc. While these models are designed to handle the large number of potential publishers and the dynamic nature of published data, enabling a straightforward query access to the global data collection, the downside is that publishers are *disintermediated* from consumers by the central site: (i) The central site has control over the visibility of publishers to user queries, and can effectively censor publishers by choosing to not index them; and (ii) The central site has complete knowledge of all information created and advertised by publishers. Even under the unrealistic assumption that the central site is trusted by publishers, the site is vulnerable to third-party censors[1] and attackers.

For this reason, we advocate a decentralized approach where there is *no* central authority, and the global data collection is *virtual*. More specifically, we make the following contributions.

1. We propose a distributed privacy-preserving publishing infrastructure in which data resides only with its owner. The infrastructure disseminates user queries to publishers, who answer them at their own discretion. Moreover, the way publishers advertise their data, in order to receive relevant queries, is designed to prevent any third party from pinpointing which publisher advertises what data (even when extensively colluding with or attacking community members).

2. Given the virtual nature of the global data collection, we address the challenging problem of efficiently disseminating queries to publishers that contain data items matching a specified query. We propose a distributed index structure, UQDT, that is organized as a union of *query dissemination trees* (QDTs), and realized on an

[1]See, for example, National Coalition Against Censorship (http://ncac.org/) and OpenNet Initiative (http://opennet.net/)

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

overlay (i.e., logical) network infrastructure. Each QDT has data publishers as its leaf nodes, and overlay network nodes as its internal nodes; each QDT internal node maintains a summary of the data advertised by publishers in its subtrees. Unlike Distributed Hash Tables (DHTs), no QDT node has complete knowledge of all the publishers that publish an advertised data item.

3. We define a notion of *publisher k-anonymity* which guarantees that for every publisher p and published data item d, the information stored in the UQDT, as well as the communication required to maintain the UQDT, are insufficient to distinguish p from $k-1$ other potential publishers of item d. We show how to configure the UQDT to guarantee publisher k-anonymity even when an arbitrary number of UQDT nodes are compromised by hacking, subpoena, collusion, or impersonation attacks.

4. The adoption of the UQDT solution hinges on its performance. We present algorithms that use the UQDT for routing queries to publishers efficiently, following the parent-child links from a QDT and making effective use of the advertised data summaries maintained by QDT internal nodes. While a single QDT suffices in principle to route queries, this results in congestion at the upper levels of the QDT, severely limiting the throughput of the overall index structure, and making it potentially vulnerable to Denial of Service attacks. We build on well known techniques for scalable dissemination trees and for "Russian Doll" search over sets [24]. We show how UQDT can achieve load balancing and throughput maximization for a workload W by a judicious combination of (i) Overlaying multiple QDTs over the network, each with a distinct root, and arranging for queries in W to be channeled in parallel through distinct QDTs, and (ii) Maintaining limited selectivity information about data items to help inform the routing strategy. To the best of our knowledge, there are no works that combine multiple trees for load balancing and hierarchical summaries for ad-hoc query routing in distributed systems.

5. We experimentally evaluate UQDT design tradeoffs through extensive simulations, using a real Wikipedia collection comprising about 1.1 million documents of total size 8.6GB. We demonstrate that UQDT can maximize throughput by preventing any overlay node from becoming a bottleneck. In addition, we show in [13] how UQDT empowers information publishers to join democratic communities and query their global collection in an ad-hoc fashion using expressive queries. To this end, we explore various QDT topologies (e.g., Scribe [6] generated multicast trees, as well as balanced structures), number of QDTs, and routing strategies (based on the selectivity information maintained), and show that (i) One can statically identify a near-optimal number of QDTs for any specified QDT topology, which maximizes throughput by preventing any overlay network node from becoming a bottleneck, and (ii) Maintaining selectivity information about a few popular data items ($2-3\%$) achieves considerable gains over random routing, and is almost as good as a "fully informed" routing strategy.

Paper Outline. We start with an overview of our proposed framework and the space of design tradeoffs in Section 2. Section 3 presents our operation choices, followed by the analysis of publisher k-anonymity in Section 4. Experimental setup and results are presented in Sections 5 and 6. We discuss related work in Section 7 and then conclude.

2. OVERVIEW OF OUR FRAMEWORK

Data and Query Model. For the purpose of information discovery and flexible querying, we abstract information as collections of *data items*, where each data item is described by a set of *content descriptors (CDs)*. CDs are an abstraction of keywords, terms, or other atomic information units [30]. For instance, in information retrieval applications, data items are text documents, and the CDs are the terms appearing in them. In relational databases, collections are tables, data items are tuples, and CDs are (attribute,value) pairs. Further examples are given in Section 5. Given a data item D, we denote its set of CDs with $\mathsf{cd}(D)$.

We consider queries expressed as sets of CDs, and denote the set of CDs of query Q with $\mathsf{cd}(Q)$. We say that data item D *matches* query Q if $\mathsf{cd}(Q) \subseteq \mathsf{cd}(D)$. Notice that the case of matching conjunctive keyword queries against text documents (the most common Information Retrieval operation) corresponds to the particular case in which CDs are keywords. Given a data collection $\mathcal{D}$, the *result* of Q on $\mathcal{D}$, denoted $Q(\mathcal{D})$, is the set of data items in $\mathcal{D}$ that match Q: $Q(\mathcal{D}) := \{D \in \mathcal{D} \mid D \text{ matches } Q\}$.

Communities of Data Publishers and Consumers. We consider communities of autonomous publishers, who join the community with their own locally stored data collection and make it available for querying. In return, they can query the *global collection* consisting of the union of all local collections.

Given our focus on providing democratic community access for autonomous publishers, we adopt a decentralized approach. In this setting there is no central control authority: the global collection is *virtual*, data resides only with the publishers owning it. The advantage is that publishers maintain complete control over who accesses their content, and how the content is "advertised" to the community. The challenge is efficient query evaluation while avoiding naive broadcast of queries to all publishers. We propose a distributed index structure that supports sending a query Q to all publishers relevant to Q while minimizing the number of irrelevant publishers reached by Q. We say that a publisher is *relevant* to Q if one of its local data items matches Q.

Our indexing solution targets a service-oriented logical network, in which we distinguish two types of nodes. There are data *publisher nodes* (community members) that provide data services and connect to the network via direct links to nodes at its edge. The data are indexed inside the network, which consists of a set of interconnected and reconfigurable *router nodes*. These are responsible for routing queries to the relevant publishers. In an internet-scale distributed setting, it is natural that routers are controlled by a multitude of distinct network providers covering different autonomous administrative domains. Thus, no single provider controls more than a fraction of the entire network, and the resulting architecture is not centralized.

While different queries might hit the same set of nodes, our goal is to balance the community search generated load at routers during query dissemination while preserving low space usage of index at a node and still preserving publisher k-anonymity.

Design Requirements. We consider the following key requirements on the infrastructure design. First, published data should not be relinquished to anyone but to community members, and only by answering queries upon successful credential authentication of the query issuer. Note that harvest-index-query methods (e.g., centralized solutions) fail. Second, publishers should advertise just enough information in the community to be reached by user queries without disclosing their identity. Publishers advertise the contents of their local store by declaring a set of CDs appearing in their local collection. Note that not all existing CDs need to be declared, especially if they pertain to private data items. The advertised information plays the role of a distributed index that is described next.

Publisher k-anonymity. We propose a notion of privacy that protects community members by preventing an attacker from associating them with the CDs they advertise. We define *publisher k-anonymity* (detailed in Section 4), which guarantees that for every publisher p and published CD d, the information stored in the

infrastructure, as well as the communication required for maintenance, are insufficient to distinguish p from $k-1$ other potential publishers of d. The distributed index guarantees publisher k-anonymity even when the UQDT nodes are compromised by hacking, subpoena, collusion, or impersonation attacks.

Query Dissemination Trees. We propose the organization of the internal nodes into a logical tree we call a *Query Dissemination Tree (QDT)*. The internal QDT nodes are routers, the publishers are leaves. Regardless of which querier initiates a query Q, Q is sent to the root of the QDT, whence it propagates down the tree to the publishers. The intention is that, when Q reaches a node n with no publishers in its subtree that are relevant to Q, n prunes its subtree from the search, i.e. it does not forward Q to its children. This pruning saves the network traffic and processing at n's descendants.

One immediate technical difficulty associated with this goal is how to instrument the index to efficiently determine that none of n's descendant publishers are relevant to Q. Of course, it is infeasible to maintain at every node n the collection of all data items in n's subtree as being prohibitively wasteful in terms of space. It would also defeat the purpose of preserving privacy of publishers: it would require a publisher p to trust every router on the path leading to p from the root. This is an unrealistic prerequisite.

We present in two steps the way routers exploit the advertised CDs by publishers. In a first cut, we assume that it is feasible to store at every node n the set $\mathsf{cd}(n)$ of all CDs declared by publishers located in n's subtree (we revisit this assumption shortly). This assumption is supported by empirical evidence that, for real data sets, the overlap of CDs across data items in a collection is considerable, and the union of all CDs (with duplicate removal) is orders of magnitude smaller than the combined size of the collection. For instance, in Section 5 we describe a collection of 1.1 million Wikipedia articles of combined size 8.6 GB that has only 3.2 million distinct CDs. Note that when only $\mathsf{cd}(n)$ is stored at a router n, n does not know which CD appears in which publisher, nor which sets of CDs appear together in a data item. This offers publishers an added degree of protection against compromised routers.

Query Routing in Single-QDT. In this setting, we consider the following simple query routing algorithm. Every query Q posed by a querier p is initially sent to the root of the QDT (in a message containing both Q and p's address). When a router node n receives the message, it forwards it in parallel to each of its children in QDT if and only if $\mathsf{cd}(Q) \subseteq \mathsf{cd}(n)$. When a publisher node is reached, it sends back to p the result of Q against its local collection. Note that when $\mathsf{cd}(Q) \not\subseteq \mathsf{cd}(n)$ holds, it is guaranteed that n's publisher descendants are irrelevant to Q. Therefore, the first-cut routing algorithm never prunes relevant publishers, thus ensuring that the final result of Q over the global collection is computed in full. In contrast, when $\mathsf{cd}(Q) \subseteq \mathsf{cd}(n)$ holds, it is not necessarily the case that at least one publisher in n's subtree is relevant to Q. This is because the CDs in $\mathsf{cd}(Q)$ may not be co-located in the same data item, or even at the same publisher. Therefore, the first-cut algorithm may forward queries unnecessarily, generating non-minimal traffic and processing load. This is a result of the unavoidable trade-off between censorship resistance and evaluation performance.

EXAMPLE 2.1. *Throughout the paper we use the following running example. Consider a network of 25 nodes that integrates news from 8 newspaper websites $P_1, ..., P_8$ (the remaining 17 nodes are routers). Figure 1(a) shows the CDs declared by each publisher (i.e., simple keywords). Consider also a query workload consisting of the four queries shown in Figure 1(b). Without showing the actual documents, assume that for every query Q there is at least one newspaper website that publishes a document matching Q.*

Assume for now that the routers and publishers are organized in the single-QDT configuration QDT_1, shown in Figure 3(a). The router nodes are identified by their preorder traversal rank. For simplicity, we assume that it is feasible for each node n to store all CDs declared by the publishers in its subtree, $\mathsf{cd}(n)$. For example, node 2 stores all CDs published by P_1 and P_2, thus $\mathsf{cd}(2) = \{$Peking, Tibet, stocks, train, money, yak tea, Hong Kong$\}$.

Publisher p	CDs declared by p, $\mathsf{cd}(p)$
P_1	Peking, Tibet, stocks, train, money
P_2	Peking, yak tea, Hong Kong, train
P_3	Peking, Tibet, yak tea, Hong Kong, money
P_4	Peking, freedom, yak tea, stocks, money
P_5	Peking, freedom, yak tea, stocks, money
P_6	freedom, Tibet, stocks, money
P_7	freedom, yak tea, stocks, money
P_8	freedom, yak tea, stocks, money

(a) Publishers' declared CDs.

Query Q	$\mathsf{cd}(Q)$	Query Q	$\mathsf{cd}(Q)$
Q_1	Peking, freedom	Q_3	train
Q_2	Tibet	Q_4	Hong Kong, money

(b) Query workload.

Figure 1: Running Example Setup

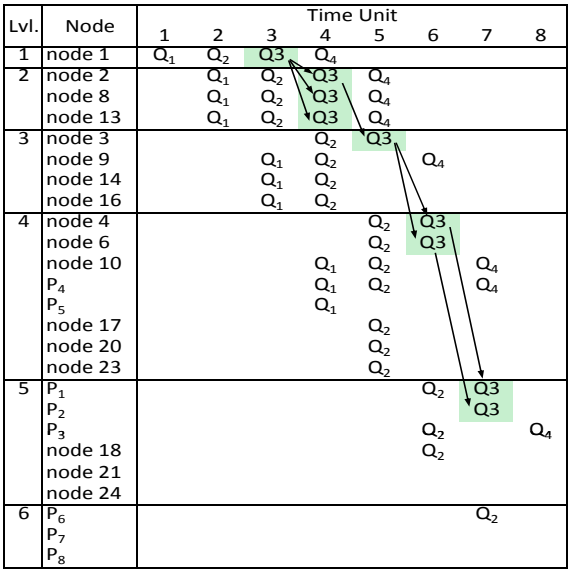


Figure 2: Query Dissemination in Single-QDT Configuration

For simplicity, let us consider in this example that every node can process exactly one query per time unit. If all queries in the workload are issued simultaneously at time 0 and processed in the order Q_1 to Q_4, then Figure 2 shows their dissemination according to the first-cut routing algorithm. For example, regardless of the issuing node, query Q_3 is disseminated in QDT_1 starting from the root node (node 1), which is congested and can only process Q_3 at time unit 3. Because train *is contained in* $\mathsf{cd}(1)$*, Q_3 is forwarded to all of node 1's children, in this case to 2, 8 and 13, where the dissemination continues recursively. Since* train *does not appear in the CD sets of nodes 8 and 13, their subtrees are pruned . However, node 2's CD set does match Q_3 and the query is routed down to node 3 at time unit 5, then to nodes 4 and 6 at time unit 6. Both these nodes have a match and Q_3 reaches the publisher nodes P_1 and P_2 at time unit 7. Each of the two publishers runs Q_3 on its local collection and sends the result back to the issuing node.* ◇

CD Set Summaries. We now revisit the assumption that all CDs in $\mathsf{cd}(n)$ are stored with every router n. We address the case when

$\mathsf{cd}(n)$ is larger than can be comfortably stored at a router n with available memory of size M. To this end, we observe that we do not necessarily need to keep the exact set $\mathsf{cd}(n)$. Instead, it suffices to store a *summary* smm^M thereof at node n. We choose to represent smm^M as a *Counting Bloom Filter* [5, 18] of size M for its well-known properties: compactness and probabilistic set membership of CDs (i.e., no false negatives, control over false positives rate).

We obtain the final version of our routing algorithm by replacing in the above first-cut every containment test with a call to a Bloom filter set membership test. Note that false positives do not affect the correctness of query evaluation.

Throughput Maximization. We have so far confined our discussion to the routing of a single query through the network. We next extend our solution to handle query workloads (sets of queries).

We start by observing that the arrival of a query at node n triggers measurable computation effort pertaining both to the processing of the query and to its forwarding to n's children. This limits the number of queries passing through n per time unit and can lead to congestion. Since queries pruned at upper levels in the tree never reach the lower levels, the fraction of any workload W reaching node n is a subset of the fraction reaching its ancestors. In particular, the root becomes a bottleneck since it is reached by all of W. In contrast, edge routers at the leaves are reached by relatively small fractions of W and may not be heavily utilized.

EXAMPLE 2.2. *Revisiting Figure 2, observe that the number of query messages reaching the nodes is significantly skewed among the tree levels, and ultimately among the nodes, decreasing from the root to leaves. Because all queries touch the top 2 levels, their nodes receive 4 messages each, while nodes on the lower levels receive 0, 1, 2 or 3 messages. Overall, it takes a total of 8 time units to disseminate all queries, of which the root alone introduces a delay of 4 time units, while nodes 21 and 24 remain idle.* ◇

We propose to alleviate congestion at the upper levels of the QDT by spreading the load more uniformly across the nodes. Currently, there are two main solutions to achieve this. One class of algorithms replicate data (or indices of it) redundantly at the router nodes. Thus, each router can initiate to answer queries. Nevertheless, this incurs increased updates cost as well additional space cost to store all replicas which is inappropriate with our initial set of goals. In contrast, we propose to partition the global data collection and interconnect the publishers for each partition block in a different overlay. We show next how this technique alleviates congestion while still preserving the space usage at routers.

Therefore, our solution consists in overlaying multiple QDTs over the network, each with a distinct root, and arranging for various fractions of W to be channeled in parallel through distinct QDTs. Since all QDTs are supported by the same underlying logical network, a network node n participates in several QDTs, receiving and forwarding queries via each of them. Balancing the load involves arranging for the distribution of levels associated with n to be (as close as possible to) uniform across the set of all QDTs. For example, the fact that n receives a high fraction of the queries flowing through QDT_1 because it resides on an upper QDT_1 level, is compensated by n being reached by only a small fraction of the queries flowing through QDT_2, where it resides on a lower level.

The goal of splitting the query workload into fractions that flow through distinct QDTs raises two fundamental technical obstacles.

The first pertains to controlling memory consumption at the router nodes. If a node n participates in multiple QDTs, it must maintain separate summaries for each of its subtrees. It is important that the total space used by the union of all summaries associated with n should not exceed the space used by n's summary in the single-QDT configuration. We satisfy this requirement by arranging for each of n's summaries to pertain to *disjoint* CD sets. To this end, we partition the space of all possible CDs into a number of k disjoint blocks $\mathcal{P} = \{B_i\}_{1 \le i \le k}$. (We discuss shortly what considerations go into picking the value of k, and we describe in Section 3 how the partitioning is achieved in practice.) We call each B_i a *CD block*. We assign to each CD block its own QDT, obtaining a family UQDT = $\{QDT_i\}_{1 \le i \le k}$.

The second problem is the query semantics preservation: we need to ensure that, by being routed only on a single QDT, a query is guaranteed not to miss any relevant publishers. We achieve this soundness property by requiring each QDT to satisfy the following:

(‡) QDT_i contains as leaves all publishers whose local data collection has at least one CD in common with B_i.

We defer to Section 3 the discussion on how the internal nodes of each QDT_i are organized.

Query Routing with Multiple QDTs. For every query Q, we pick the QDT to send it to as follows. The partition $\mathcal{P}$ induces a partition $\mathcal{P}_Q = \{Q_j\}_{1 \le j \le m}$ on $\mathsf{cd}(Q)$, such that for each $Q_j \in \mathcal{P}_Q$ there is $B_i \in \mathcal{P}$ with $Q_j = \mathsf{cd}(Q) \cap B_i$. We call each such Q_j a *query block* and we say that the CD block B_i *corresponds* to Q_j. Note that by definition each query block corresponds to precisely one CD block, which in turn corresponds by construction to precisely one QDT. Given a query block $Q_j \in \mathcal{P}_Q$, we can therefore refer to "the" corresponding QDT, and denote it with $\mathsf{qdt}(Q_j)$.

In general, Q has $1 \le m \le |\mathsf{cd}(Q)|$ query blocks, with corresponding QDTs $\mathsf{qdt}(Q_1), \ldots, \mathsf{qdt}(Q_m)$. For routing Q, we only pick one of these QDTs, say $\mathsf{qdt}(Q_j)$. Regardless of how this pick is taken, we send to the root of this QDT a message containing three components: (Q_j, Q, p), where p is the address of the initiating querier. $\mathsf{qdt}(Q_j)$ routes this message as described above in the single-QDT case, with only three minor refinements:

- since every internal node n can participate in various QDTs, n stores one summary smm^M_T per QDT T;
- n uses Q_j for routing in $\mathsf{qdt}(Q_j)$ (i.e. for lookup into the summary $n.\mathsf{smm}^M_{\mathsf{qdt}(Q_j)}$); and
- leaf nodes use Q for evaluation against their local data.

EXAMPLE 2.3. *Example 2.2 shows how the congestion appears inevitably in the upper levels of the dissemination tree. Here, we show how congestion can be alleviated by using multiple QDT overlays over the same nodes.*

We consider a configuration of 4 QDTs, each corresponding to a block in the CD space partition $\mathcal{P}$. $\mathcal{P}$ is shown in Table 1.

Block	CDs
B_1	Peking, freedom
B_2	Tibet, yak tea
B_3	Hong Kong, stocks
B_4	train, money

Table 1: Blocks of the 4-Partition

In general, internal nodes can be connected in any configuration at the network overlay layer. Figure 3 depicts 4 possible QDTs, one per CD space partition block.

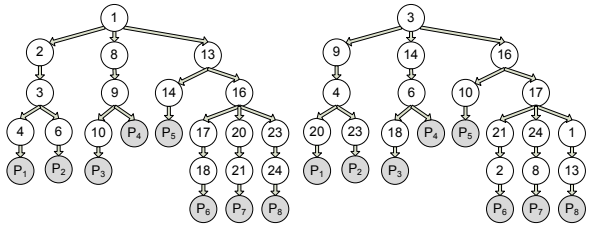
Table 2 shows the CD summaries maintained at every router. Since a router appears in multiple QDTs, it actually manages a set of summaries. For simplification purposes, we assume that each summary stores the exact set of CDs rather than its approximation.

Figure 4 presents the routing diagram in the 4-partition over time, where queries $Q_1..Q_4$ are issued simultaneously at time 0.

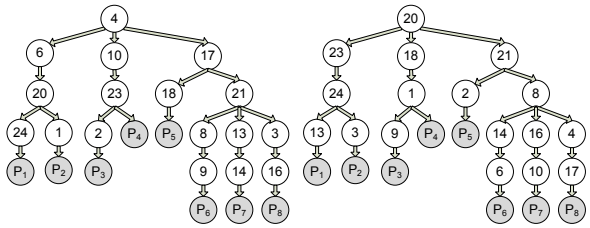
Query Q_1 is a conjunctive query both of whose CDs fall in the first partition block B_1. The only routing choice is hence the tree corresponding to B_1, namely QDT_1 shown in Figure 3(a). Since $\mathcal{P}$'s blocks are disjoint, single-conjunct queries also have only one

Node	Tree	Data summary
4, 6, 3, 2, 10	QDT_1	Peking
9, 8, 14, 13, 1		Peking, freedom
18, 17, 21, 20, 24, 23, 16		freedom
20, 2, 21	QDT_2	Tibet
23, 10, 8, 24, 13, 1		yak tea
4, 9, 18, 6, 14, 17, 16, 3		Tibet, yak tea
24, 18, 9, 8, 14, 13, 16, 3, 21, 17	QDT_3	stocks
1, 2		Hong Kong
20, 6, 23, 10, 4		Hong Kong, stocks
9, 1, 18, 2, 6, 14, 10, 16, 17, 4, 8, 21	QDT_4	money
3		train
13, 24, 23, 20		train, money

Table 2: CD Summaries in the 4-QDT Configuration



(a) QDT_1 for B_1. (b) QDT_2 for B_2.



(c) QDT_3 for B_3. (d) QDT_4 for B_4.

Figure 3: Query Distribution Trees for the 4-Partition

routing choice. For instance, Q_2 and Q_3 are routed using QDT_2 in Figure 3(b), respectively QDT_4 in Figure 3(d). Q_3's routing on QDT_4 by CD train *is highlighted in Figure 4.*

In contrast, query Q_4 intersects CD blocks B_3 and B_4, which induce two query blocks: $\mathcal{P}_{Q_4} = \{\{\text{Hong Kong}\}, \{\text{money}\}\}$. This offers two routing alternatives: either by using CD Hong Kong *on QDT_3, or by using* money *on QDT_4. In the diagram, we assume that QDT_3 was picked. When the subquery hits publishers P_2 and P_3 the full query Q_4 is tested on the local store (only P_3 has a match for both CDs of Q_4).*

Comparing with Example 2.1, notice that the 4-QDT configuration outperforms the single-QDT case: the former takes 6 time units to complete the dissemination, while the latter needs 8. The improved throughput is due to better load balance: contrast the behavior of routers 21 and 24, which remain completely idle in Figure 2 but shoulder part of the dissemination task in Figure 4.

Finally, observe that the benefit of better node utilization outweighs the drawback of using query blocks for pruning, instead of the entire (and more selective) set of query CDs. Indeed, the 4-QDT configuration wins despite its less aggressive pruning which leads to slightly more messages (50, as opposed to 46 for one QDT). ◇

It is easy to check that property (‡) implies the soundness of our query evaluation algorithm:

PROPOSITION 1. *For every query Q, partition $\mathcal{P}$, and pick of j, our query routing algorithm correctly computes Q's answer.*

Obviously, for single-block queries there is no choice and the QDT is uniquely determined. However, in the general case of multiple-block queries, Proposition 1 uncovers an optimization opportunity: the judicious QDT choice (out of several equally sound alternatives) towards throughput maximization. So we treat the spectrum of possible routing strategies as an optimization dimension in its own right.

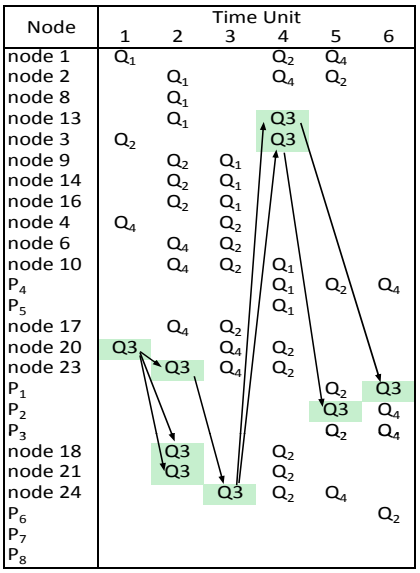


Figure 4: Query Dissemination in 4-QDT Configuration

The UQDT Design Space Layout. We remark that the number k of blocks in the partition $\mathcal{P}$ of the CD space defines a spectrum of possible configurations of the same network, thus adding a new dimension to the optimization space. One extreme of this spectrum is the case $k = 1$, which we have discussed above as the single-QDT configuration. At the other extreme, we have the case in which each block of $\mathcal{P}$ is a singleton CD. We refer to it as the *per-CD* configuration. We argue next that neither of the extremes results in optimal throughput, and that the value of k is an optimization dimension we need to explore. Indeed, Example 2.1 and Example 2.3 show that the single-QDT configuration is certainly not optimal, being outperformed by a 4-QDT configuration for the given query load. At the same time, constructing too many QDTs is counterproductive, since the increase in k decreases the size of the query blocks, thus resulting in less selective lookups in each node's summary. This translates into less pruning, i.e. more query forwarding messages: the 4-QDT configuration in Example 2.3 generates 50 messages, as opposed to the 46 of the single-QDT configuration in Example 2.1. In conclusion, as k increases, we observe two opposite effects: an increase in load balancing potential, but also in the overall load (number of messages) in the network. An independent consideration that precludes extremely high values of k is that the maintenance of any overlay network involves a small, but non-zero control traffic overhead [6]. Maintaining too many QDTs would amplify this overhead.

In Section 3, we discuss the following optimization issues, all of which have significant impact on query throughput: How can a partition $\mathcal{P}$ of the infinite space of all possible CDs be chosen and represented finitely (this includes determining the value for k)? How can $\mathcal{P}$ be used to efficiently determine $\mathcal{P}_Q$? How are the various QDTs corresponding to $\mathcal{P}$ organized for better throughput? How does the choice of QDT (the pick of j) impact throughput?

3. OUR APPROACH

In Section 2, we have provided an overview of our proposed solution for query dissemination, identifying the dimensions of the space of possible implementations. We delegate to Section 4 the

discussion of how to configure and maintain the UQDT to ensure publisher anonymity. As a proof of concept for the viability of our approach, we developed an actual implementation, described in this section and evaluated experimentally in Section 6.

QDT Topology. There are many possible topologies for organizing the router nodes into a QDT. Although our solution is generic, we investigate two approaches.

First, we take the pragmatic approach of "piggy-backing" on top of a mature overlay tree-building approach to disseminate messages to groups of nodes (also known as *multicast* groups). Since multicast overlay trees are constructed with a different goal than QDTs, it is not immediately clear that they are optimal for query dissemination (though we show experimentally that we can "convert" them, achieving very good performance). However, one advantage of delegating the QDT construction to such off-the-shelf technology is that it is equipped to exploit information on the topology of the underlay network with minimal control overhead. Moreover, it maintains overlays dynamically, adapting to the change in underlay network conditions. One widely-used representative of this class of tools is Scribe [6], and FreePastry [32] is one popular open-source implementation, which we used.

In addition, we consider home-grown QDTs built for the express purpose of balancing the forwarding effort among the routers. Since every router forwards a query to each of its children, the forwarding effort is linear in the node's fanout. This suggests constructing (nearly) balanced QDTs, with as little variation as possible in the node fanouts. We need to construct such trees ourselves, since Scribe does not guarantee balanced trees.

QDT Maintenance. When a publisher p joins the community, it declares a set $\mathsf{cd}(p)$ of CDs it is willing to answer queries about. Recall from Section 2 that, to preserve soundness of query evaluation, we must satisfy property (‡). To this end, we determine (as described shortly) all the CD blocks with non-empty intersection with $\mathsf{cd}(p)$, which in turn lets us identify all QDTs that p must join. The act of joining a given QDT is taken care of by Scribe which identifies the router node that will become the new publisher's parent. Once the publisher is added to QDT T, the CD summaries of all its ancestors in T are updated by inserting $\mathsf{cd}(p)$ into them. This insertion is implemented by simply obtaining once and for all the set of indices $\mathsf{ind}(\mathsf{cd}(p))$, which is then passed bottom-up from p to T's root, so that every router on the way can increment its corresponding Bloom filter counters. When p leaves a QDT T, the index set $\mathsf{ind}(\mathsf{cd}(p))$ is also sent bottom up to p's ancestors in T, each decrementing the corresponding counters. The case when an existing publisher p changes its list $\mathsf{cd}(p)$ of declared CDs leads to the propagation of similar counter operations.

Partitioning the CD Space. An important issue we need to address is how to represent the partition $\mathcal{P}$ of the CD space finitely, and how to efficiently determine which block a given CD belongs to. As described above, we need this test to quickly identify the QDTs a new publisher must join. Moreover, the same test is required to compute the induced partition $\mathcal{P}_Q$ of a query Q, in order to identify the QDT candidates for routing Q. We describe here our solution assuming that we have already established the number k of blocks in $\mathcal{P}$ (we discuss below how we determine k with an eye on load balancing). Given k, we implement $\mathcal{P}$ simply as a hash function $h_{\mathcal{P}}$ from CDs to the set $\{1, \ldots, k\}$, where $h_{\mathcal{P}}$ distributes CDs uniformly over its range. Then each block $B_i \in \mathcal{P}$ consists of all CDs mapped by $h_{\mathcal{P}}$ to i: $B_i := \{d \mid d \text{ is a CD }, \ h_{\mathcal{P}}(d) = i\}$. Of course, each CD block is potentially infinite so we never really materialize it. Indeed, we don't need to: all we need is to quickly determine, given a CD d, which CD block it belongs to. This operation is implemented as a constant-time invocation of $h_{\mathcal{P}}(d)$.

Load Balancing. The way we determine the number k of QDT trees, as well as their actual construction, are motivated by the goal of spreading the load evenly across routers. In the following, we denote with N_r the number of router nodes in the service provider's overlay network, and with N_p the number of publisher nodes. Since in any QDT T, every router node is reached by a larger fraction of the query flow through T than its descendants in T, we need to ensure that for every router n, the distribution of QDT levels n resides at is close to being uniform. We adopt a solution which is certainly not the only possible one, nor necessarily optimal, but it is easy to implement and (as proven experimentally in Section 6) it yields excellent performance. We start by constructing (using Scribe) a single QDT T_1 whose internal nodes are the N_r routers and whose leaves are the N_p publishers. Scribe tends to build trees of low height, in which the root has a significant fanout that dominates the fanouts of nodes in lower levels. The root and its children receive by far the highest fraction of queries flowing through the tree, and are hence in most need of relief through load balancing.

Denoting with N_u the number of nodes on the top 2 upper levels in T_1 ($N_u = 1+$ number of router children of the root), we construct $k = \lfloor \frac{N_r}{N_u} \rfloor$ QDTs, $\{T_i\}_{1 \le i \le k}$. Each T_i is an isomorphic copy of T_1, whose nodes are obtained by keeping the same N_p leaves and only re-shuffling the N_r internal nodes as follows. To completely specify T_i, we show how its N_r internal node positions are populated with the actual N_r routers. This can be formalized as a function a_i from the set of N_r routers to the set $\{0, \ldots, N_r - 1\}$ of positions in T_1. We adopt the convention that the *position* of node n corresponds to n's rank in the breadth-first, left-to-right traversal of T_1 (position 0 is the root). Let $\pi(n) := (n - N_u) \mod N_r$ be the right-to-left cyclic permutation with step N_u on $\{0, \ldots, N_r - 1\}$. If a_1 specifies the initial QDT T_1, then for each $1 < i \le k$, we populate T_i by cyclically permuting with step N_u the nodes of T_1 a total of $i - 1$ times: $a_i := \pi^{i-1} \circ a_1$.

EXAMPLE 3.1. *In Example 2.1, there are $N_r = 17$ routers, and the root of the initial tree QDT$_1$ has three children, yielding $N_u = 4$. We compute $k = \lfloor \frac{17}{4} \rfloor = 4$ and construct the 4 trees in Figure 3. Notice that the trees in Figure 3(b), 3(c), 3(d) are obtained by cyclically permuting to the left by 4 steps the tree in Figure 3(a) once, twice, respectively three times.* ◇

It is easy to see that our methods of determining the number of QDTs, and of populating them, ensures the following fairness property: *all routers appear precisely once in the top 2 levels of any QDT.* Furthermore, the k level values associated to every router are distributed almost uniformly over all possible level values in T_1. For instance, in Figure 3, router 1 appears on levels $1, 4, 4, 3$.

Finally, note that building $k + 1$ QDTs actually degrades the load balance, because the additional cyclic permutation causes a "wrap-around" that returns some of the routers residing on the top two levels in T_1 to the top two levels of T_{k+1}, subjecting these routers to unfair load (since we use the floor function to determine k, the wrap-around is not necessarily complete). In general it follows that, to maximize balance, we want to use a number of QDTs that is a multiple of $\lfloor \frac{N_u}{N_r} \rfloor$. In Section 6, we validate this rule experimentally, also showing that choosing multiples higher than 1 is unnecessary: they do not improve load balance, while leading to higher control overhead.

Routing Strategies. We next discuss how a node n that initiates a query Q picks the QDT to route Q on. First, n uses the hash function $h_{\mathcal{P}}$ described above to compute $\mathcal{P}_Q = \{Q_j\}_{1 \le j \le m}$, which in turn determines the set of candidate QDTs $\{\mathsf{qdt}(Q_j)\}_{1 \le j \le m}$. If $m > 1$, n picks one of these candidates. We consider several alternatives for implementing this pick.

A simple solution is to choose $1 \le j \le m$ at random, in the hope

that randomness avoids sending many queries down the same QDT and alleviates congestion. We call this the *random* routing strategy.

We also consider alternative strategies, all attempting to alleviate the effect we discussed in Section 2: as the number of QDTs increases, the selectivity of query blocks decreases (recall that, when routing Q through QDT $\mathtt{qdt}(Q_j)$, only the CDs in Q_j are looked up in the summaries). This results in increased overall query forwarding and processing in the network. To compensate for this effect, the routing strategy should ideally use the most selective query block Q_j for routing, as this results in the most aggressive pruning of $\mathtt{qdt}(Q_j)$'s subtrees during Q's dissemination.

Identifying the most selective block of a query is not trivial, as it requires determining the frequency of every CD in the global collection, and storing these statistics (or making them otherwise accessible) at every publisher. We call the strategy assuming each publisher's access to this information the *fully-informed* routing strategy. Assuming independence between the CDs, the publisher initiating Q computes the selectivity of a query block Q_j as the product of the individual frequencies of the CDs in Q_j. Fully-informed routing is very expensive in terms of both space and traffic. Indeed, for large global collections, the number of CDs can be considerable. Moreover, space consumption is exacerbated by the fact that the frequency must be stored with every potential query initiator. A more serious problem is the traffic arising because the global collection is virtual: gathering and maintaining the appropriate statistics requires constant communication between nodes.

We therefore investigate a less ambitious strategy: instead of identifying the most selective query block for Q, its initiator p only tries to avoid using the least selective ones. It suffices to this end to maintain and store at each publisher a short list of the s least selective (most frequent) CDs in the global collection, with s a relatively small value ensuring small storage space and maintenance traffic consumption. Finding the overall top s most frequent CDs amounts to the distributed top-s heavy hitters estimation [2, 28].

We implement a simple solution that exploits the already existing QDT overlays, employing them in a dual role as multicast (data dissemination) trees. With every CD they advertise, publishers declare its frequency in their local collection. Each node n maintains a list $n.L$ of length at most s entries, each containing a CD and its frequency. For non-root routers, the list gives the s most popular CDs across all their QDT subtrees. For QDT roots and publishers, the list holds most popular s CDs across the global collection. Whenever a node n updates its list, it propagates the new list bottom-up along all QDTs n participates in. If n is a root, it propagates its list to the other $k-1$ roots. Whenever the root of a QDT T updates its list, it disseminates it top-down to all publishers in T.

When node n issues a query Q, it picks the QDT corresponding to Q's most selective block according to the information in $n.L$. Note that some query blocks may contain CDs not occurring in the $n.L$ list. These are treated as selective CDs, and blocks with the highest number of selective CDs are preferred. If multiple such query blocks exist, n breaks the tie by computing the selectivity of the conjunction of popular CDs in each block, using $n.L$. If this still leaves more than one candidate query block, one is picked at random. We call this strategy *partially-informed* routing, and observe that it leads to a spectrum of strategies parameterized by the size of internal state reserved for the list of popular CDs. We use the term *x-informed routing* in short for partially-informed routing based on the list of the most popular $x\%$ of CDs. Notice that 100-informed routing becomes fully-informed, and 0-informed routing degenerates to random routing. In Section 6, we show experimentally that, by keeping track of even very short lists, we observe performance very close to the fully-informed strategy, and much better than the random strategy.

EXAMPLE 3.2. *We revisit Example 2.3, explaining why query Q_4, which had two routing alternatives, was sent to QDT$_3$. To enable fully-informed or partially-informed routing, publishers maintain frequencies of (some of) the CDs in the global collection, which in our case include* money *(published by 7 publishers),* stocks *and* yak tea *(published by 6 publishers). Notice that* Hong Kong *is declared by only 2 publishers and hence more selective than* money*, which is why it is preferred by the fully-informed routing strategy. Since CD* Hong Kong *appears in block B_3, the corresponding tree QDT$_3$ is used. The same outcome is achieved for partially-informed routing, assuming for instance that publishers maintain only the 3 most popular CDs: the list includes CD* money*, signaling to Q_4's initiator to avoid routing by it.* ◇

Finally, when no selectivity information is available, we fall back on *heuristic* routing: simply direct Q to the QDT corresponding to one of Q's maximum-cardinality blocks, breaking ties with random picks. This strategy is based on the heuristic that higher numbers of conjuncts tend to yield higher selectivity.

4. PUBLISHER K-ANONYMITY

The challenge for the design of the UQDT maintenance protocol is to simultaneously guarantee that (i) queries reach all relevant publishers, (ii) network traffic is minimized and congestion avoided, and (iii) publishers are encouraged to register with the dissemination infrastructure, being guaranteed that the registration will not expose their connection to certain sensitive CDs. We have shown above how the UQDT infrastructure addresses requirements (i) and (ii). In this section, we focus on item (iii).

The privacy guarantee: publisher k-anonymity. Our approach here is to adapt the notion of k-anonymity from relational table anonymization [33]. In our context, we wish to guarantee that for every publisher p and every CD c, if p advertises c, then the routing information stored in the UQDT and exchanged during UQDT maintenance does not allow p to be distinguished from at least $k-1$ other potential publishers of c. This involves ensuring that the set of publishers connected to the same edge router consists of at least k members, and that even edge routers cannot tell which among its $k+$ publishers advertises any given CD. This latter requirement defends against the event when edge routers are compromised (by hacking, subpoena, or impersonation). As described shortly, this guarantee involves collaborative computation among the publishers of an edge router. We describe how this collaboration can be conducted without exposing a publisher even if (i) all other publishers in its group have been compromised and are colluding against it but the edge router is trusted, or (ii) the edge router and up to $N-k$ publishers have been compromised, where N is the number of publishers in p's group.

First observe that, if every edge router e could be trusted to behave as prescribed in Section 3, and to never be compromised, then the publishers would remain k-anonymous if ancestors of the edge routers were compromised. Indeed, recall that e only stores and communicates to its parents in the UQDT the Bloom filter summary of $\mathtt{cd}(e)$, i.e. the union of all CD sets advertised by its publishers. The Bloom filter, which is the only exposed information, does not record *which* of the $k+$ publishers advertises a particular CD, nor which sets of CDs occur together in some document. $\mathtt{cd}(e)$ is therefore insufficient to pinpoint who among e's publishers advertises any given CD. By ensuring that e's subtree contains sufficiently many publishers advertising CDs as a group, we enable each publisher to remain anonymous by "hiding in the crowd" comprised of this group. As an added bonus, the CD summary implementation is

hash-based and does not distinguish among two distinct CDs with the same hash code. Publishers exploit this by declaring the hash codes of their advertised CDs rather than their actual value. An inspection of the edge router's summary will therefore fail to answer with certainty even the simple question whether a given CD is advertised by some publisher, let alone by a given publisher. Recall from Section 3 that the price traded off for this added protection is that false positives to CD membership tests lead to queries being forwarded unnecessarily to the publishers, thus affecting performance. Our experiments show that this overhead is small and the false positives are negligible. Also note that, the further up an ancestor a of an edge router e is, the more fuzziness a's CD summary will contain. If a is compromised and its summary exposed, then each of e's publishers is hidden not only in the crowd of e's $k+$ publishers, but in the larger crowd of all publishers in a's subtree. Finally, note that the publishers not in a's subtrees are not affected.

But how do we defend against the case when the edge router e itself is compromised? Since the collection of documents stored at publishers is dynamic, every one of e's publishers p (in some QDT_i) needs to declare to e the set of CDs it wants to advertise (or stop advertising). A compromised e would record this information if p were to declare its CDs directly. To preserve p's anonymity even against e, we designed the following protocol.

Publishers $p_1, .., p_N$ (with $N \geq k$) of the same group participating in QDT_i declare to their edge router e only *batch updates* of their advertised CDs, instead of sending up individual updates. To advertise a new set of CDs, publisher p_j installs them in an initially empty Bloom filter. That is, it starts from a filter with all counters set to 0, hashes each CD, and increments the counters whose index is given by the hash codes. The resulting filter is publisher p_j's update U_j. The batch update S is the vector sum of all publisher updates, $S = U_1 + \ldots + U_N$. e receives S and sums it to its CD summary (since both are represented as Bloom filters, the operation reduces to vector sum). CD deletions are handled by subtracting S from e's summary. It is easily shown that this protocol supports the correct maintenance of CD summaries. More, it ensures that e cannot figure out the individual updates, as it only receives their sum over all publishers. Note that e doesn't even see the actual CDs; it obtains only their count (muddled by hash collisions). We can show that this protocol preserves k-anonymity even if all routers are compromised.

We must address one last issue: where can the batch update S be computed? Asking e to do so would defeat the purpose, as it would involve each publisher to send its update to e. Instead, S is computed collaboratively by the publishers, without involving e. To find out the nodes connected to e, individual publishers can publish/broadcast the router they are connected to (e.g., by running one's own Web service answering the "buddies" request using public key cryptography to rule out impersonation). To defend even against the case when publishers themselves are compromised, we use the classical cryptographic technique of secure multi-party computation [22]. This allows a set of N publishers to compute the batch update without revealing the individual values to each other or to outside observers of their communication traffic. This shields every publisher even against the case that all other publishers in its group are colluding against it, assuming that they are not also colluding with e. If e as well as some some publishers are colluding, then e knows the updates of these publishers and can subtract them from the overall batch update, retrieving the batch update of the uncompromised publishers. If fewer than k of these remain, then anonymity decreases. One can defend against this case by arranging sufficiently large publisher group sizes N, so that compromising more than $N - k$ of them is practically infeasible. Another possible defense consists in publishers joining UQDT only together $k - 1$ trusted "buddies". This does require trust, which however is bounded and does not need to extend to a vast unknown infrastructure.

Secure multi-party computation involves overhead. However, note that the publisher group only needs to compute as many sums as entries in the Bloom filter vector. This is a constant of the UQDT, independent of the size of the global document collection. Further, the computation is performed only on batch updates, so its overhead is manageable by adjusting the update frequency.

Finally, recall k-anonymity guarantee holds on a per-CD basis. Since UQDT partitions the CD space among its member QDTs, there is no interaction between QDTs to derive compromising information. If the guarantee holds for each QDT in isolation, it holds for entire UQDT.

What we do not defend against. We emphasize that we are only concerned with putting publishers' minds at ease w.r.t. the safety of participating in the UQDT. We do not address here the orthogonal problem of how publishers decide whether to answer a query once it reaches them (recall that the query answer is sent directly to the query issuer), or whether to identify themselves in the answer. To guarantee that the query issuer is not an impersonator, and that the query answer cannot be observed by third parties, one can adopt existing techniques based on authentication credentials, encrypted channel communication, and anonymization proxies (discussed in related work). We do not aim to make the infrastructure impervious to large-scale censoring attacks, such as a governmental agency completely shutting down the Internet in a region, or a denial-of-service (DOS) attack overloading the UQDT to decrease data availability. However, note that the effect of DOS attacks is mitigated by our load balancing scheme, which maximizes throughput.

5. EXPERIMENTAL SETUP

The Initial Overlay Network. To analyze the effects of our implementation choices on query dissemination, we built a simulator of a 10,000-node overlay network consisting of $N_p = 9,400$ publisher and $N_r = 600$ router nodes.

A Real Data Set. To obtain true-to-life community, we simulate a distributed community that shares an XML dump of Wikipedia, comprising about 1.1 million real Wikipedia documents which amount to 8.6 GB [14]. We simulate that documents are each brought into the community by one of the 9,400 publishers. Due to lack of information on which publisher generated which document, we assign the documents to publishers in a uniform random manner.

CD Definition. Since Wikipedia uses structural schema (i.e., not ontological), the majority of the tags on the root-to-leaf XML paths are concerned with document organization, providing no semantic meaning. This motivates us to consider CDs defined as pairs (t, w), where w is a keyword that appears in context t, given by the last XML tag on the path from the root to w. We include this tag to support context-aware queries that go beyond standard keyword search. Moreover, we focus only on the tags that carry meaning to users (e.g., *"link", "b", "title", "subtitle" and "category"*). The combination of such CDs yields a complex set of about 3.2 million distinct CDs accounting for 24% of the set of all distinct CDs obtained by considering all possible tags. We have tried other CD definitions (see [13]) and obtained analogous results. The point is that the flexibility of CD definition is a key enabler for striking the right balance between query expressivity and space overhead.

EXAMPLE 5.1. *We setup the Bloom filter for each node's summary as follows. Fixing the false positive rate at 10^{-2}, it follows from [18] that the optimum number of hash functions is 7 when the size of the Bloom filter at every router (assuming a single-QDT*

configuration and counters of size 1 bit) is $M = 3.6$ MB, which represents only 0.044% of the global collection size. For larger counter sizes, the false positive rate is even lower. For k QDTs, the global memory consumption per node stays the same, since the k Bloom filters at every node summarize disjoint sets of CDs. Each Bloom filter has size $3.6/k$ MB, and the same error rate of 10^{-2}. $\diamond$

Query Workload. We force the dissemination process to work under two extreme query types. We construct a family of 10 workloads $\{W_c^F\}_{1\leq c\leq 10}$, each consisting of $5,000$ c-conjunct queries drawn at random from the space of queries with no match against the global collection. Similarly, we build the family of workloads $\{W_c^T\}_{1\leq c\leq 10}$, each comprising $5,000$ c-conjunct queries drawn at random from the space of queries with at least one match in the collection. We also generate the $50,000$-query workload $W^T = \bigcup_{c=1}^{10} W_c^T$. The $\{W_c^T\}_i$ workloads increase the overall forwarding effort by forcing QDTs to send queries all the way to (some) leaves.

Scribe QDTs. Recall from Section 3 that, even in multiple-QDT configurations, the QDTs are isomorphic. We obtain a (unique up to isomorphism) QDT^S topology using Scribe. We first convince ourselves of the faithfulness of the simulation, by generating a family of 20 Scribe tree topologies for the same node set (by varying the order in which the nodes join the network). We observe only non-essential variations across the family, thus boosting our confidence that picking any tree in this family is representative of Scribe's behavior. The particular Scribe tree we pick has $9,400$ leaf nodes and 600 internal nodes, 5 levels, average fanout of 16.7, and a maximum fanout of 101. The fanout features a very skewed distribution, decreasing from root to leaves (this holds for all 20 Scribe trees we considered). The distribution of the number of nodes per tree level 1 to 5 is as follows: 1 node (the root), 40 (of which 3 are publishers), $1,189$, $6,163$ and $2,607$ nodes. We determine the number k of isomorphic copies as in Section 3. We have $N_r = 600$ routers in total; among the 40 children of the root, 37 are routers. We obtain $N_u = 1 + 37 = 38$ and hence $k = \lfloor \frac{N_r}{N_u} \rfloor = \lfloor \frac{600}{38} \rfloor = 15$.

Fanout-balanced QDTs. We extend our simulation to QDT topologies not created by Scribe. We consider a topology QDT^B that uses the same router and publisher nodes, but eliminates the skewed fanout distribution that is typical of Scribe trees. This is beneficial since a node's fanout influences its forwarding cost. We first organize the 600 routers into a balanced skeleton tree with fanout 8, where levels $1, 2, 3, 4, 5$ have, respectively, $1, 8, 64, 512$ and the remaining 15 nodes. Next, we connect the $9,400$ publishers to this skeleton tree, achieving for each node a fanout of 16 or 17. There are 75 non-leaf routers in the skeleton tree, and each receives 8 publishers, for a total fanout of 16. Among the leaf routers in the skeleton tree, 400 receive 17 publishers and 125 receive 16 publishers. We determine the number k of fanout-balanced QDTs in the usual manner: $k = \lfloor \frac{N_r}{N_u} \rfloor = \lfloor \frac{600}{9} \rfloor = 66$.

Metrics. Our goal is to improve the query throughput of the multi-QDT overlay, defined as the number of queries answered per time unit. Throughput is a manifestation of two more fundamental factors, namely the processing and forwarding effort at every router. For a given workload W, we define the *processing load* at node n, $PLoad_W(n)$, as the number of query messages reaching n across all QDTs it participates in. The *forwarding load* at n, $FLoad_W(n)$, is the number of query messages leaving n along all QDTs it participates in. Notice that none of the two measures is derivable from the other, since $FLoad_W(n)$ depends on n's fanout distribution (over the QDTs it participates in) and on the amount of pruning at n. For both load flavors, we define the *peak* load, which is the maximum load over all nodes. Clearly, decreasing either or both kinds of peak load results in increased throughput.

EXAMPLE 5.2. *In Example 2.1, 46 messages are used to disseminate 4 queries in 8 time units, while in Example 2.3, 50 messages disseminate the same query workload in 6 time units. Defining throughput as the number of queries answered per time unit, the 4-QDT case has the higher throughput. The reason we don't simply use throughput as a metric is that it requires assumptions on the relative duration of processing and forwarding cost (in our running example, we take the simplifying assumption that forwarding cost takes constant time, independent of fanout).*

In Figure 2, the processing load for a node is the number of queries on its row. For example, the processing load for node 13 is 4, which is also the peak processing load. In Figure 4, the peak processing load is 3, experienced for instance by nodes 2 and 10.

The forwarding load can be read by inspecting the transitions between columns and keeping track of parent-child relationships in the various trees. In the single-QDT case (Figure 2), root node 1 has the highest peak forwarding load, 12 (it forwards each of the 4 queries to its 3 children). In the 4-QDT configuration (Figure 4), the peak forwarding load is 6 messages, experienced by node 20 (1 message for Q_2, 3 for Q_3 and 2 for Q_4).

Notice that, compared to the single QDT, the 4-QDT configuration decreases both processing and forwarding peak load, which leads to improved throughput regardless of the concrete values of the per-query processing and forwarding cost. $\diamond$

The above considerations suggest comparing configurations by their degree of reduction of the peak processing and forwarding loads. Note, the ideal load balance is achieved when the peak "drops" to the average load, which is measured as the average over all router nodes.

Note that our goal is not merely to achieve balance, as one can do so without improving throughput by simply raising the average load. Indeed, as discussed in Section 2, with increasing number k of QDTs both kinds of average load increase (though only slightly, as shown experimentally). This is because routing by smaller query blocks results in less pruning, which increases the overall number of messages. Thus, the lowest possible average load occurs in the single-QDT configuration, and represents the ideal target for lowering the peak load. Since we are interested in closing the gap between the peak load in a k-QDT configuration and the ideal peak load, we report the *ideal-to-actual load ratio* metric, defined as the ratio between the peak load in the k-QDT and the average load in the single-QDT configuration.

6. SIMULATION RESULTS

In this section, we explore through extensive simulations the space of configurations defined by the three dimensions given by the topology of QDTs, number of QDTs, and routing strategies. We confirm empirically that the configuration choices we advocate achieve near-optimal peak load reduction, and therefore near-optimal throughput.

Warm-up: Single-QDT Configuration. In this experiment, we confirm that the number of messages reaching the various levels in a single-QDT configuration is sufficiently skewed to justify our load-balancing efforts, in particular that the routers on the first two levels of the tree bear the brunt of the load. For query workloads W_2^F and W_2^T, and the Scribe topology QDT^S, we report in Table 3 for every level the total and average number of messages seen by its nodes. Notice that the average number of messages per node decreases drastically below the upper two levels. Also notice that, unsurprisingly, workload W_2^T generates more overall messages, since its matching queries undergo less pruning than those in W_2^F.

Effect of Number of QDTs. In this experiment, we validate our method for determining the number k of QDTs (recall Sec-

QDT level	W_2^F		W_2^T	
	# msg. per level	Avg. # msg. per node	# msg per level	Avg. # msg. per node
1	5,000	5,000	5,000	5,000
2	200,000	5,000	200,000	5,000
3	173,066	146	636,507	535
4	28,509	5	513,464	83
5	4,869	2	193,575	74
Total	411,444	-	1,548,546	-

Table 3: Messages per Level ($k = 1$, QDT^S, fully-informed)

tion 3). For workload W^T and fully-informed routing, we increase the number k of QDT^S copies from 1 to 31. Figure 5 shows the average and the peak load for both processing and forwarding.

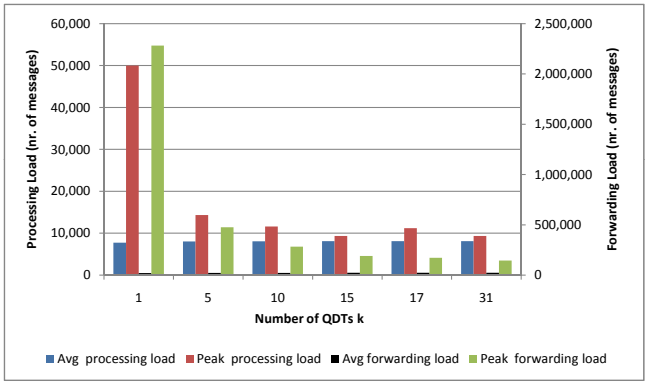


Figure 5: Effect of Number of QDTs (W^T, QDT^S, fully-informed routing)

Notice that with increasing k, the gap between the peak load and the average load decreases considerably. The highest load imbalance occurs for $k = 1$ as shown in the big gap between the peak and the average values for both the processing and the forwarding load. As predicted by our analysis in Section 3, $k = 15$ is indeed the "sweet spot" where the minimum gap is measured. Increasing k to 17 increases this gap. This is because the two additional cyclic permutations cause a "wrap-around" of the routers from the top two levels of QDT$_1^S$ to the top two levels of QDT$_{16}^S$ and QDT$_{17}^S$ and thus introduce load imbalance. Also note that there is no point in looking at strict multiples of $\lfloor \frac{N_r}{N_q} \rfloor$ beyond $k = 15$, as they cost more overlay maintenance overhead without bringing the peak load any closer to the average load.

Finally, we observe that the negative effect of increasing overall number of messages with increasing k does occur: the average processing load is indeed the lowest for $k = 1$ since routing is done using with all conjuncts, thus benefiting from maximum routing selectivity. However, the increase is very slow when compared to the decrease in peak load. The negative effect of average load increase is outweighed by that of peak load reduction, as shown by the closing gap between peak and average loads.

We observe this behavior more accurately in terms of the ideal-to-actual peak load ratio, which for increasing k approaches the ideal value 1. For example, the ideal-to-actual peak load ratio for the same values of k as in Figure 5 are, respectively: 6.42, 1.85, 1.49, 1.21, 1.44 and 1.20.

Figure 5 shows the same trend for the forwarding load, with the only difference that, while the gap of peak and average loads decreases with growing $k \leq 15$ and saturates once k exceeds 15, we remain far from the ideal reduction (for which the ideal-to-actual load ratio is 1). This is explained by the forwarding load's correlation with the node fanouts and the fact that Scribe builds trees with highly skewed fanout distribution.

Effect of Static Load Indicators. In the extended version [12], we introduce two load indicators to capture statically the balance degree of a k-QDT configuration, and we confirm experimentally a good correlation with the dynamic query dissemination performance. These indicators measure for each node n its average tree level, and its average fanout (over all QDTs it participates in).

Effect of Routing Strategy. We next compare the routing strategies defined in Section 3. For the partially-informed strategy, we consider the case when publishers maintain the top s popular CDs for s = 43k, 74k and 124k, corresponding respectively to 1.37%, 2.33% and 3.89% of the total number of CDs in the global collection. We compare the strategies for workload W^T and QDT^S, reporting the ideal-to-actual peak load ratio in Figure 6.

First, we note that random routing performs worst, closely followed by heuristic routing. Both strategies are significantly outperformed by the (partially- or fully-)informed ones for every $k > 1$ (with the exception of $k = 1$ when all routing strategies coincide).

The family of informed routing strategies follows a common trend: with increasing $k \leq 15$, the gap between ideal and actual load shrinks drastically, reaches the sweet spot at $k = 15$ and essentially saturates for $k > 15$ (with a slight increase at $k = 17$ for processing load, due to discussed imbalance introduced by the wrap-around).

Interestingly, random and heuristic routing behave slightly differently: at $k = 5$, the actual load gets closer to the ideal load than at $k = 15$. This is caused by the following effect. The more QDTs, the more query blocks, which decreases the chance of a random pick hitting the most selective block. With increasing k, this effect starts generating non-minimal traffic, eventually canceling the load balancing effect. This explains why the random strategy degrades with increasing k. The reason the degradation saturates is that the number of query blocks cannot increase indefinitely (it must saturate once all blocks become singletons). Heuristic routing suffers from essentially the same problem: the more blocks we split a query into, the smaller the variation in block cardinality. Recall that, for same-cardinality query blocks, heuristic routing degenerates to random. In contrast, for the informed routing family, the experiments show that this negative effect remains subtle, being canceled out by the judicious choice of selective query blocks.

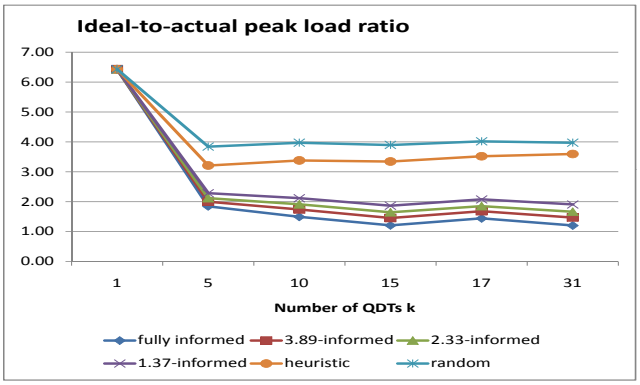
Finally, we get very close to the benefits of fully-informed routing in negligible space overhead, by maintaining the frequency for only a small fraction (3.89%) of all CDs. These results strongly recommend partially-informed routing over the other strategies.

Effect of QDT Topology. We repeated all above experiments using the fanout-balanced QDT topology QDT^B, observing the same trends as for the Scribe topology QDT^S. We do not report the detailed results for lack of space. Instead, we summarize in Table 4 the comparison between the Scribe-generated and the fanout-balanced topology, relative to the peak load reduction. For query workload W^T and the fully-informed routing strategy, we show the ideal-to-actual peak load ratio factor for the appropriate number of QDTs (15 for QDT^S and 66 for QDT^B).

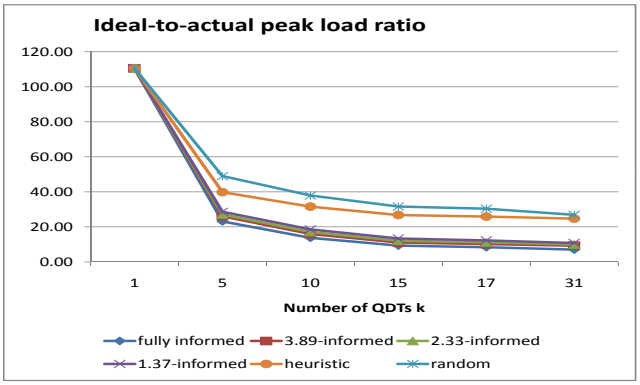
ideal-to-actual peak load ratio	Scribe (QDT^S) $k = 15$	fanout-balanced (QDT^B) $k = 66$
processing	1.21	1.18
forwarding	9.3	2.3

Table 4: Effect of QDT Topology (W^T, fully-informed)

Notice that both topologies come within reach of the ideal load reduction (when the ideal-to-actual load ratio is 1) for processing load. However, for forwarding load the Scribe topology misses the ideal by an order of magnitude, whereas the fanout-balanced topology only by a factor of 2.3. The main reason not even the



(a) Processing load.



(b) Forwarding load.

Figure 6: Effect of Routing Strategy (W^T, QDT^S)

QDT^B topology reaches the ideal forwarding load reduction is the inherent imbalance between the number of routers and publishers: the perfect configuration consists of a perfectly balanced tree whose internal nodes are routers and whose leaves are publishers. We did not simulate such a configuration because in practice we have no control over the numbers of routers and publishers.

Our experiments confirm that fanout-balanced topologies result in improved forwarding load reduction over Scribe topologies without sacrificing processing load reduction. The benefit of using Scribe is of logistic nature, as it comes off-the-shelf with the overlay maintenance functionality. An advantage of our solution is its generality; it assumes no control over the shape of the QDT, focusing on extracting the performance inherent in the topology.

7. RELATED WORK

To provide high throughput and scalable search over distributed content we identify three research directions related to our work, mainly, mediation based, replication based, and partitioning based solutions. We analyze each of them in terms of efficiency, query power expressivity and publisher anonymity.

Mediation approach. In the mediator approach the data residing with different publishers in the network is collected and accessed via a single site, also called the mediator. This architecture is the standard for most of the current search engines and online hosted communities. Our focus on publisher anonymity makes the centralized architecture less than ideal, since publishers need to trust the mediator when registering their data with it (or when allowing crawlers to collect their data). Moreover, the central point of access to data is vulnerable to censorship attacks (governments have been known to press search engines to not return query answers on certain political hot topics, and to turn over their records).

Query dissemination in P2P networks. Recently there has been a large body of work that focuses on finding only the peers with relevant data to a user's query. These methods construct data summaries at nodes and use them as routing indices (e.g. [11, 9]) to disseminate the query in the network toward the relevant publishers. These works are not focused on publisher anonymity.

Replication based approaches. One way to increase data availability and to balance the load, and therefore to improve the system throughput is to replicate all or parts of the data (or indices of it) redundantly at the router nodes [27, 20]. Disseminating queries to publishers in such a scenario is simple since each such router has global information. This however means that compromising a single router will violate the anonymity of all publishers.

Partitioning-based approaches. An alternative way to leverage the distributed computational power is based on partitioning the data across the peers. The publishers and the consumers do not need to know the details of the partitioning scheme to send data or queries. The network takes care of identifying the relevant matching data to the queries. Our approach is partition based.

DHT based. A partition-based solution to building routing indices that is popular among structured P2P networks is to leverage distributed hash tables (DHTs). A DHT provides a distributed logical abstraction of object identifier lookups (e.g. filename lookups) over the physical underlay [20, 39, 1, 4, 23, 34]. A follow-up body of work builds hierarchies of overlays based on DHTs. To improve locality, the hierarchies are created based on the document content similarities [40] or on the nodes proximity in the network to minimize latency [38, 29, 21]. However, DHTs are inappropriate for the problem we study, since DHT nodes maintain *complete* knowledge of all the publishers that advertise specific data items. An attacker can gain global publisher information for specific CDs by simply compromising the responsible DHT node. In contrast, no UQDT node maintains knowledge about the publishers of any data item.

Other routing strategies. Our load balancing technique applies to any tree topology, and is complementary to research on determining the best topology for dissemination (e.g. see [25, 10] for tree-shaped P2P indices that are not DHT-based).

Koloniari and Pitoura [26] consider the problem of routing path queries over schema-less XML documents in a P2P system. Their approach is similar to our single-QDT configuration, with the attendant limitations, and our technique for maximizing throughput has the potential to be useful for this problem as well. At the opposite spectrum, [19] builds a QDT for each published data item which can sometimes be impractical due to the large number of CDs. Both these related works do not address the publisher anonymity issue.

P2P publish/subscribe. A complementary problem is that of distributed publish/subscribe, wherein query subscriptions from users are maintained in a distributed index structure, and data items are disseminated to subscribers when they are published. Various multicast techniques are used for dissemination [3, 6]. Although constructed for a different goal, we show how off-the-self multicast Scribe trees for data dissemination can be used for QDTs.

Content-based publish/subscribe approaches match the entire published content against (possibly aggregated) subscription queries. A good example is ONYX [15], wherein a dissemination tree is rooted at each publisher. Each router maintains for each interface an aggregate subscription (XML query) that summarizes all the subscriptions downstream along that interface. A published data item starts from the root (the publisher), and gets forwarded to downstream interfaces whose corresponding aggregate subscriptions match the data item. Chand and Felber [7] take a similar approach. SemCast [31] aggregates subscriptions in a centralized way using a cost-based model.

Censorship resistance. Most of the existing censorship-resistant systems including Free Haven [17], Publius [36], and Tangler [35] are based on anonymizing the communication, and therefore anonymizing the end-to-end communicating entities (Tor [16], Freenet [8], etc.) This is usually done by using proxy based services with DHTs (e.g., Anonymizer.com), or using servers to encrypt and route the traffic through established anonymous tunnels over the other nodes. Note that both DHTs and encryption/routing servers need to be trusted by publishers. These anonymization techniques can be used as a complementary measure together with UQDTs. The advantage of UQDTs is that they do not require publishers to trust them.

8. CONCLUSION

The dawn of the age of online communities poses the challenge of empowering information publishers to join democratic communities and query their global data collection in an ad-hoc fashion. We present an infrastructure that meets this challenge by allowing data to reside with its owners and by supporting queries against all data collection with no need for any trusted central authority, which disintermediates publishers from consumers. These queries are evaluated by dissemination to relevant publishers using a distributed index structure. Our solution precludes third parties from learning the exact publisher–CD associations, guaranteeing publisher k-anonymity (i.e. for every CD, there are at least k possible publishers) even if nodes of the dissemination index, or peer publishers are compromised.

Technically, our approach is dual to the conventional work on data dissemination. Our contributions towards efficient query dissemination range from identifying the design space (with its trade-off dimensions, relevant metrics, and notion of optimality), to introducing solutions that achieve near-optimality with low overhead.

Partially-informed routing emerges as the best-value strategy, with low space overhead to yield the same benefits as fully-informed routing. The solution exploits crucially the dual role of QDTs, deploying them as both query and statistics dissemination trees. While we show that fanout-balanced topologies are closest to optimal, an advantage of our solution is its generality, in that it focuses on extracting the performance inherent in any given topology.

9. REFERENCES

[1] S. Abiteboul, I. Dar, R. Pop, G. Vasile, D. Vodislav, and N. Preda. Large scale P2P distribution of open-source software. In *VLDB* (Demo), 2007.

[2] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, 2003.

[3] G. Banavar, T. Chandra, B. Mukherjee, N. Nagarajarao, R. Strom, and D. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS*, 1999.

[4] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Minerva: Collaborative P2P search. In *VLDB* (Demo), 2005.

[5] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. Commun. ACM, 13 (7), 1970.

[6] M. Castro, P. Druschel, A. M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE JSAC*, 2002.

[7] R. Chand and P.A. Felber. A scalable protocol for content-based routing in overlay networks. In *Symposium on Network Computing and Applications*, 2003.

[8] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A Distributed anonymous information storage and retrieval system. In *LNCS*, 2001.

[9] G. Conforti, G. Ghelli, P. Manghi, and C. Sartiani. Scalable query dissemination in XPeer. In *IDEAS*, 2007.

[10] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. P-Ring: An efficient and robust P2P range index structure. In *SIGMOD*, 2007.

[11] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS*, 2002.

[12] E. Curtmola, A. Deutsch, K.K. Ramakrishnan, and D. Srivastava. Censorship-resistant Publishing. In *Technical Report CS2010, UC San Diego*, March 2010. http://db.ucsd.edu/xtreenet/xtreenetTR10.pdf

[13] E. Curtmola, A. Deutsch, D. Logothetis, K.K. Ramakrishnan, D. Srivastava, and K. Yocum. XTreeNet: Democratic Community Search. In *VLDB* (Demo), 2008.

[14] L. Denoyer and P. Gallinari. Wikipedia XML corpus. In *SIGIR*, 2006.

[15] Y. Diao, S. Rizvi, and M. Franklin. Towards an internet-scale XML dissemination service. In *VLDB*, 2004.

[16] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX*, 2004.

[17] R. Dingledine, M.J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *International Workshop on Designing privacy enhancing technologies*, 2001.

[18] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: A scalable wide-area web cache sharing protocol. In *IEEE Transactions on Networking*, 8(3), 2000.

[19] W. Fenner, M. Rabinovich, K. K. Ramakrishnan, D. Srivastava and Y. Zhang. XTreeNet: Scalable overlay networks for XML content dissemination and querying. In WCW, 2005.

[20] L. Galanis, Y. Wang, S.R. Jeffery, D.J. DeWitt. Locating data sources in large distributed systems. In *VLDB*, 2003

[21] P. Ganesan, K. Gummadi, and H. Garcia-Molina. Canon in G Major: Designing DHTs with hierarchical structure. In *ICDCS*, 2004.

[22] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *ACM Conference on Theory of computing*, 1987.

[23] M. Harren, J.M. Hellerstein, R. Huebsch, B.T. Loo, S. Shenker, and I. Stoica. Complex queries in DHT-based Peer-to-Peer networks. In *IPTPS*, 2002.

[24] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *VLDB*, 1995.

[25] H. Jagadish, B.C. Ooi, K.L. Tan, Q.H. Vu, and R. Zhang. BATON*: Speeding up search in peer-to-peer networks with a multi-way tree structure. In *SIGMOD*, 2006.

[26] G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In *EDBT*, 2004.

[27] D. Lomet. Replicated indexes for distributed data. In *PDIS*, 1996.

[28] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE*, 2005.

[29] A. Mislove and P. Druschel. Providing administrative control and autonomy in peer-to-peer overlays. In *IPTPS*, 2004.

[30] M. Ott, L. French, R. Mago, and D. Makwana. XML-based semantic multicast routing: An overlay network architecture for future information services. In *Globecom* 2004.

[31] O. Papaemmanouil and U. Cetintemel. SemCast: Semantic multicast for content-based data dissemination. In *ICDE*, 2005.

[32] Pastry. A substrate for peer-to-peer applications. http://freepastry.org.

[33] L. Sweeney. K-anonymity: A model for protecting privacy. In *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10 (5)*, 2002.

[34] P. Triantafillou and T. Pitoura. Towards a unifying framework for complex query processing over structured Peer-to-Peer data networks. In *DBISP2P*, 2003.

[35] M. Waldman and D. Mazieres. Tangler: A censorship-resistant publishing system based on document entanglements. In *CCS*, 2001.

[36] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *USENIX*, 2000.

[37] World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Full-Text. Working draft. http://www.w3.org/TR/xquery-full-text/.

[38] Z .Xu, R. Min, and Y. Hu. HIERAS: a DHT based hierarchical P2P routing algorithm. In *ICPP*, 2003.

[39] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *SIGCOMM*, 2004.

[40] S. Zoels, M. Eichhorn, A. Tarlano, and W. Kellerer. Content-based hierarchies in DHT-based Peer-to-Peer systems. In *SAINT-W*, 2006.