Scalable Summary Based Retrieval in P2P Networks

Wolfgang Müller, Martin Eisenhardt, and Andreas Henrich
Chair of Media Informatics
Bamberg University
wolfgang.mueller@wiai.uni-bamberg.de, martin.eisenhardt@wiai.uni-bamberg.de,
andreas.henrich@wiai.uni-bamberg.de

ABSTRACT

Much of the present P2P-IR literature is focused on distributed indexing structures. Within this paper, we present an approach based on the replication of peer data summaries via rumor spreading and multicast in a structured overlay.

We will describe Rumorama, a P2P framework for similarity queries inspired by GlOSS and CORI and their P2P-adaptation, PlanetP. Rumorama achieves a hierarchization of PlanetP-like summary-based P2P-IR networks. In a Rumorama network, each peer views the network as a small PlanetP network with connections to peers that see other small PlanetP networks. One important aspect is that each peer can choose the size of the PlanetP network it wants to see according to its local processing power and bandwidth. Even in this adaptive environment, Rumorama manages to process a query such that the *summary* of each peer is *considered* exactly once in a network without churn. However, the actual number of *peers* to be *contacted* for a query is a small fraction of the total number of peers in the network.

Within this article, we present the Rumorama base protocol, as well as experiments demonstrating the scalability and viability of the approach under churn.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search Processes; H.3.5 [**Online Information Services**]: Data Sharing

General Terms

Algorithm, Performance, Reliability

Keywords

P2P, Similarity Search, Distributed IR

1. INTRODUCTION

Peer-to-Peer (P2P) networks are groups of independently administered computers that share resources in order to achieve a goal common to its units, the peers. In the last few years, efficient search in P2P networks has been the object of considerable research. The community has advanced from simple broadcast-based approaches to search (often called *flooding search*) towards efficient search for large integers [12] and real-valued vectors [11], as well as information retrieval [14, 2] or queries in distributed RDF repositories based on structured broadcast and super-peers [10].

The present article gives and evaluates a contribution to the processing of similarity queries in P2P networks.

In the current literature about complex queries in P2P networks, there are the following main directions: *Firstly*, there is work about distributed indexing structures for similarity search (*e.g.* [14]). These structures work very much like local indexing structures. Some structure invariant is maintained, and indexing data to be inserted into the data structure is put at the right place in the resulting data structure. In distributed indexing structures, this involves shipping the indexing data to the place it belongs. *Secondly*, there are networks that keep most of the indexing data in the peer it originated from, and that interpret search as a *routing* problem, *i.e.* routing the query to the peers that contain the searched data (*e.g.* [2]).

Distributed indexing structures become interesting by their conceptual similarity to centralized indexing structures. Initially (before some efficiency-enhancing simplifications are made) the results obtained from distributed indexing structures correspond precisely to those obtained from centralized indexing structures. The main drawback of distributed indexing structures, and particularly that of inverted files is the fact that for very large networks the data shipped for processing a query exceeds the bandwidth available [7]. Another important drawback of distributed indexing structures is that participating peers have to ship their indexing data in order to become searchable. Especially for peers that share big amounts of data, this can cause huge communication cost for participating in the network. The third aspect is the fact that processing of leaving peers becomes important, as in distributed inverted files the indexing data does not reside on the same peer as the indexed data. So, a peer leaving the network takes some indexing data for data residing on live peers out of the network. Conversely, some indexing data belonging to the leaving peer is staying within the network. This calls for redundancy of indexing data and has to our knowledge been acknowledged as an existing, but solvable problem in P2P networks with churn.

Issues with excessive use of bandwidth for query process-

ing in distributed inverted files are being countered by hybrid approaches [13, 5].

In contrast to distributed indexing structures, peers in summary-based systems distribute only a *summary* of the data residing on them to other peers. So the cost for becoming *searchable* by the network is much smaller than the corresponding cost in distributed indexing structures. The small cost encourages peers to contribute to the network. The summary data actually distributed in the network is very small with respect to the full indexing data, giving ample room for replication of such data, favoring robustness. In contrast to distributed indexing structures, indexing data and indexed data reside on the same peer, thus leaving peers take their indexing data *and* the corresponding indexed data with them out of the network.

Within this paper we focus on a summary-based approach that is inspired by multi-collection distributed IR such as GlOSS and CORI [1, 4] sketched in the following.

1.1 GlOSS: Limiting Cost in Distributed IR

The GlOSS (*Gl*ossary *O*f *S*ervers *S*erver) is a server that knows the location of a set S of information retrieval servers. In addition to the location, it knows a short summary of each server's data. When receiving a query, the GlOSServer[1] will use the summaries in order to select a subset C of S ($C \subset S$) containing the servers which are most likely to store useful results. Then, the query is forwarded to the servers in C. The GlOSServer will collect the results obtained and present the combined results to its user. Obviously in the ideal case, the GlOSServer will contact only servers containing the most relevant items. The degree of success obtainable from GlOSS and similar methods depends on the suitable combination of the algorithm for choosing the servers and the quality of the summaries (see *e.g.* [1, 4]). In [9] we have presented and evaluated two summary representations that can be employed for distributed content based image retrieval.

1.2 PlanetP: GlOSS in Small P2P Networks

GlOSS and similar methods are not applicable in the setting of P2P networks with churn, *i.e.* peers entering and leaving: GlOSS depends on the GlOSServer being online all the time and being strong enough to forward all the incoming requests. However, such assumptions cannot be made in P2P networks where peers (*i.e.* servers) can come and go at any time.

PlanetP [2] brings GlOSS to a setting of moderately-sized P2P networks. It does so by replicating the summaries. In a stable network, each peer knows the summaries of all other peers in the network. When a new peer p_n enters the network, it will ask its first contact for all summaries it knows, and it will send its own contact information. With this, p_n is introduced to the network. All peers introduced to the network exchange – at fixed time intervals – newly obtained summaries with some randomly picked peers in the network. Using this method, summaries of new peers are spread over the whole network without having to go through the pains and cost of announcing each new peer via a broadcast.

While PlanetP appears to work well for moderately-sized networks with little churn, PlanetP is not scalable. This is due to two reasons: *Firstly,* the number of summaries each peer has to store grows linearly with the number of peers in the network, eventually exceeding the amount of memory a peer is willing to sacrifice for participation in the P2P network. *Secondly,* in a network with churn the amount of new summaries each peer has to receive in a given time frame also grows linearly with the number of peers in the network, eventually exceeding the bandwidth of most participating peers.

[1]In the following, we will write GlOSS when writing about GlOSS and similar methods, and GlOSServer when we mean an actual Glossary of Servers Server.

1.3 Rumorama: Scalable PlanetP

In the following, we will describe Rumorama, a protocol that achieves a hierarchization of PlanetP-like networks. In a Rumorama network, each peer sees the network as a small PlanetP network with connections to other peers that see other small PlanetP networks. One important aspect is that each peer can choose the size of the PlanetP network it wants to see according to its local processing power and bandwidth. Even in this adaptive environment, Rumorama manages to process a query such that the summary of each peer is considered exactly once in a stable network.[2]

We will use the term *leaf net* to denote the small PlanetP network viewed by a peer. The size of the leaf net a peer sees determines the number of summaries to be sent, received and stored by that peer. We will call p_b a *friend* of p_a if p_a potentially asks p_b for summaries. The *leaf net* of p_a is the set of friends of p_a. Rumorama enables each peer to keep its number of friends approximately constant with respect to the total number of peers in the Rumorama network ($O(1)$). Please note that the friendship relation is not symmetric, as peers can choose the size of their leaf nets individually.

For the purpose of query processing, each peer keeps a table of *neighbors, i.e.* nodes that are not within the peer's leaf net but which are useful for reaching peers outside its leaf net. As we will illustrate, the number of neighbors stored by each peer grows logarithmically ($O(\log N)$) with the total number of peers N in the Rumorama network.

It is noteworthy that there is no need for super-peers in Rumorama. Depending on the situation, all peers can (and will) act as an inner node or as part of a leaf net in the hierarchization formed by Rumorama.

2. RUMORAMA: THE BASIC IDEA

Fig. 1 depicts the situation of a peer in a Rumorama network using two dimensions for visualization. As usual, each peer is identified by an ID represented as a binary number. An ID is mapped onto a location in the 2-dimensional space by interleaving: The first, the third, the fifth, ... bit of the ID form the coordinate value in the first dimension and the second, the forth, the sixth, ... bit form the coordinate value in the second dimension.

The ID of the peer used as the querier in the following example is `[01011000...]`. This peer is located in a leaf net (indicated by the grey rectangle with a dashed border) and it maintains compact peer data summaries for all 11 peers in that leaf net. With all peers in the leaf net the peer

[2]Note that considering a summary does not mean that the peer that issued the summary has to be visited. Considering the summary of a peer is a prerequisite to including the peer's data into the query result. A consequence of churn is that fewer than all summaries are considered (see our measurements).

shares the first three bits [010] of its ID. More precisely, each peer maintains a so-called *friends mask* representing the prefix of the ID which is equal for all peers in the leaf net. In our case, the friends mask is [010]. Each peer can decide the length of his friends mask on his own. Thereby, it can maintain an upper bound for the number of friends.

In order to communicate with peers outside its leaf net, our peer maintains additional information about simple neighbors. In contrast to friends in the leaf net, the peer does not maintain peer data summaries for its neighbors. Our example peer has (at least) two neighbors for each bit in its friends mask. It has a neighbor with an ID starting with 0 and it has a neighbor with an ID starting with 1 for the first bit. For the second bit it has a neighbor with an ID starting with 00 and it has a neighbor with an ID starting with 01. For the third bit it has a neighbor with an ID starting with 010 and it has a neighbor with an ID starting with 011[3]. These neighbors are also indicated in Fig. 1.[4]

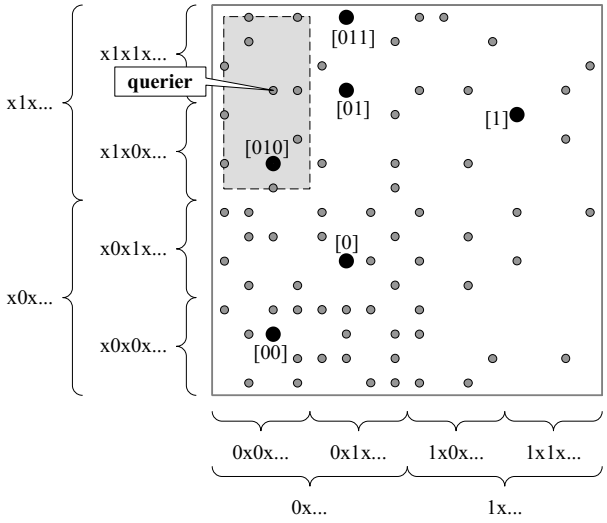

Figure 1: A peer's view of the network (the peers in the shaded rectangle are its friends and the peers indicated by the black circles are its neighbors).

When a peer wants to issue a query it is important to distribute the query to leaf nets covering the whole P2P network. In these leaf nets the query has to be processed as a local PlanetP query and the results have to be merged. To distribute the query, the querier forwards the query to its neighbors with an ID starting with 0 and with an ID starting with 1 (this is visualized by the dashed arrows [0] and [1] originating from the querier in Fig. 2). In addition, the query is augmented with a parameter denoting the length of the prefix already covered in the distribution process (1 for the first step). The two peers contacted in that way check if the length of the prefix already covered is equal to or longer than the length of their friends mask. In that case, a leaf net is reached. Otherwise, the query is forwarded to the neighbors maintained by these peers on the next level. In our example, for both peers leaf nets are not yet reached and therefore the query is again forwarded to these peers' neighbors for prefix length two. Note that this second step is processed in parallel on the two peers. For the peers reached with the prefixes [11], [10] and [01] leaf nets are now reached. Only the peer reached for the prefix [00] has to forward the query one more time.

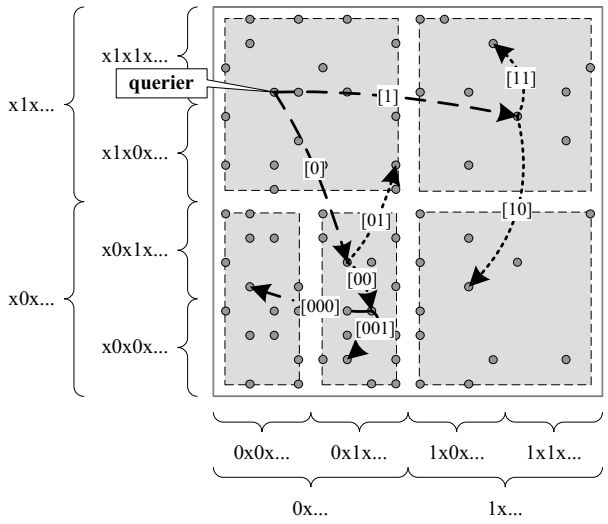

Figure 2: Distribution of a similarity query to the leaf nets in Rumorama

Now five leaf nets are reached. For each leaf net the path taken by the query is represented by the prefix given in squared brackets in Fig. 2. We call this prefix the *query mask*. The peers reached in the single leaf nets will now process the query locally in their leaf nets. If the friends mask of one of these peers is equal to the query mask, it will include all its friends in the local query processing. If the friends mask is shorter than the query mask it will involve only those friends into the query processing for which the query mask is a prefix of their ID (please note that the grey regions given in Fig. 2 actually represent these subsets of the peers' leaf nets). This way the query processing considers a disjoint and exhaustive partitioning of the P2P network.

If the querier is interested in the k best matches, the k best matches are calculated for each leaf net separately. Thereafter, the results obtained are merged following the paths on which the query was distributed in reverse order.

In the leaf nets all optimization techniques known from PlanetP can be applied. In particular it is not necessary to contact all peers in the leaf nets. Therefore, our approach guarantees that all peer data summaries are considered during the query processing. However, only a small portion of the peers is contacted.

Within the next section, we will show how the Rumorama protocol assures that each peer knows the neighbors and summaries needed for such query processing. Each peer keeps a redundant set of neighbors, thus assuring sufficient connectivity. We also show how Rumorama efficiently adapts to increasing and decreasing network size.

[3]Note that neighbors for mask [010] and [011] obviously are also neighbors for mask [01].

[4]Note that it is necessary in the concrete implementation to maintain some redundant neighbors in order to keep the system fit for work in a volatile network.

3. A CLOSER LOOK AT THE PROTOCOL

3.1 Structural Invariant & Query Algorithm

The Rumorama protocol creates a robust hierarchization of PlanetP-like networks. PlanetP uses summaries in order to reduce the number of peers needed for processing a query. The motivation for creating a hierarchization of PlanetP is the insight that it is too expensive to have all peers collect summaries of all other peers in the network. As a consequence of the hierarchization Rumorama peers have two classes of adjacent peers, *friends* and *neighbors*. A peer knows of every *neighbor* the peer `id()` (a unique 160 bit string), as well as the IP address and its UDP port. In addition to that it knows of its *friends* also the summaries of the data contained in them.

A Rumorama peer maintains a set of *neighbors* that grows *logarithmically* with the size of the network. It maintains a set of *friends* (and their summaries) that stays approximately constant with varying network size.

In order to achieve this, the Rumorama peer has to choose its friends and neighbors. As it is done in many DHT systems such as for instance Chord [12], a peer chooses its neighbors based on its own ID and other peers' IDs.

In addition to its ID, every Rumorama peer p has a *friends mask*. This friend mask is a prefix of the peer's ID `id(`p`)`. That is, by definition a peer matches its own friend mask. The friend mask determines which peers are *friends* of p. Friends of p are peers that match the friend mask `mask(`p`)` of p. The friend mask `mask(`p_a`)` of a peer p_a is defined to match a peer p_b iff `mask(`p_a`)` is a prefix of `id(`p_b`)`.

By appending bits to the friend mask or by shortening the friend mask, p_a can choose its number of friends. In a network with N nodes, a mask with length ℓ will match approximately $\frac{N}{2^\ell}$ peers.[5]

In a stable Rumorama network, every peer has chosen its preferred number of friends. In regular intervals each peer p_a asks some randomly chosen friends for peer addresses they have newly learned of more recently than a time `newSince`. When by this way, p_a discovers addresses of new friends, it asks some of its friends for the corresponding summaries. In an object oriented notation we write this asynchronous remote method invocation (RMI) as follows:

```
someFriend.requestNewFriends(mask(p_a),newSince)
```

Each peer asked for a set of friends returns the fresh friends (along with timestamps) it knows plus its own address with a current timestamp. By this, it will be spread through the network when the peer `someFriend` was last alive.

The summary retrieval message would be accordingly

```
someFriend.requestSummaries(listOfAddresses)
```

By this rumor spreading process it is assured that all peers obtain information about all their friends. In particular it is assured that no peer gets unsolicited information about other peers, *i.e.* no information about peers that do not match its friend mask.

In addition to information about their friends, peers carry information about their *neighbors*. Neighbors enable a peer to reach other leaf nets. Similar to information about its friends, a peer updates its neighbor information using a rumor spreading process. The following RMI asks for `size` randomly chosen addresses for a given mask `[x]` from a neighbor whose ID matches mask `[x]`:

```
someNeigbor[x].requestRandomNeighborsForMask([x],size)
```

Typically, a peer will maintain a small constant number (*e.g.* 10) of neighbors for each needed mask. Which neighbor masks are needed by a peer is governed by the fact that the summary of *every* peer should be considered *exactly once* in the query evaluation process. We say: we want to *reach* all summaries with a query.

Which neighbors are actually necessary is given by the query algorithm and can easily be derived from Rumorama's way of adapting to changing network size described in section 3.2.

From the example given in section 2 follows an algorithm for query processing in Rumorama networks of arbitrary size, in which the peers have arbitrary-sized friend masks: A peer p_a with friend mask `[f]`, receiving the query with query mask `[x]`

$$p_a\texttt{.query([x],query,size)}$$

will distinguish two cases:

1. The query mask `[x]` is at least as long as `[f]`. This means that p_a contains all summaries of peers matching `[f]`. This means, p_a will process the query alone *i.e.* it will act as a GlOSServer using the subset of summaries it has stored matching `[x]`. Then p_a will give back the results thus obtained to the query issuer.

2. `[x]` is shorter than `[f]`. In this case, p_a does not know all summaries necessary for processing the query. It will choose for the query masks `[x0]` and `[x1]` one neighbor each ($p_{\texttt{[x0]}}$ and $p_{\texttt{[x1]}}$) and forward the query to them, adjusting the query mask:

   $$p_{\texttt{[x0]}}\texttt{.query([x0],query,size)}$$
   $$p_{\texttt{[x1]}}\texttt{.query([x1],query,size)}$$

   $p_{\texttt{[x0]}}$ and $p_{\texttt{[x1]}}$ will process the query the same way as p_a. After processing, they will forward results to p_a. Then p_a will merge these results and will return them to the peer from which it has obtained the query.

The robustness of Rumorama is due to the fact that $p_{\texttt{[x0]}}$ and $p_{\texttt{[x1]}}$ are selected from *the set* of neighbors of p_a: If we assume failure of $p_{\texttt{[x0]}}$, p_a can choose from its neighbors another peer as forwarding target, *e.g.* $p'_{\texttt{[x0]}}$ matching the mask `[x0]`.

3.2 Adapting to Varying Network Size

Given a peer p_a with friend mask `[x]` and a network growing so strongly that $n_{\texttt{[x]}}$, the number of nodes in the network matching `[x]`, is larger than $n_{split}(p_a)$. Here $n_{split}(p_a)$ is the maximum number of summaries that p_a is willing to store. In this situation p_a will seek to lower the number of stored summaries. This can be done in two steps:

1. Append a bit to the friend mask: p_a will append a bit `b` to its mask `[x]`, such that its ID `id(`p_a`)` matches `[xb]` (= `[x]` with appended bit `b`).

[5]Under the assumption that IDs are uniformly distributed in the keyspace. This is the same assumption Chord and similar systems make.

2. Some friends will become neighbors: p_a forgets the summaries of peers that do not match the new friend mask [xb]. They—or at least a randomly chosen subset of them—become neighbors matching mask [x$\bar{\text{b}}$].

We call this sequence of operations the *split of a leaf net*.

A sketch of proof of that a split does not modify the number of summaries reached in a query process is given in [8].

From the splitting algorithm, we can deduce which neighbors p_a needs to maintain in order to be able to reach all peers within the network. Assume p_a had the id [011100101...] and the mask []. Splitting the leaf net once, it would have mask [0] and neighbors for mask [1] splitting it again, it would have friends for [01] and neighbors for [1], [00] and so forth. Neighbors for mask [0] are friends for [01] or neighbors for [00], so we do not need to *explicitly* maintain a set of neighbors for [0]. At any mask length, p_a needs to maintain neighbors for exactly the same masks as it would have if its current friends mask was the result of a sequence of splits starting at mask [].

We just described how peers react when their leaf net grows above a certain size. A peer p_a can react in a similar fashion if the number of summaries it knows shrinks below a value chosen by p_a, $n_{merge}(p_a)$.

Given p_a with mask [x1]. In this case, p_a will shorten its friend mask to [x]. p_a already knows the summaries of all peers matching [x1]. In order to learn the summaries of peers matching [x0], it will ask a neighbor matching [x0] for all summaries matching [x0]. Either this neighbor is aware of all these summaries, because its friend mask is equal to or shorter than [x0], or this peer can determine the desired summaries be asking his neighbors for masks [x00] and [x01] (which proceed in the same way) and combine the results.

3.3 Joining and Leaving the Network

For joining the network, p_n with id(p_n) needs to know the address of one Rumorama peer, say, p_a. To be fully integrated, p_n needs to obtain from p_a information about neighbors and friends. However, it is improbable that the IDs of p_n and p_a are so similar for p_n to be a friend of p_a. So, p_n has to find its place in the network, in particular its friends and neighbors. We present an algorithm that takes time logarithmic in the total number of peers in the network: To this end, p_n will retrieve from p_a its neighbors matching masks [0] and [1]. At the same time it will tell p_a its $n_{split}(p_n)$.

$$p_a\texttt{.getNeighborsOrFriendsForMasks}(\{\texttt{[0]}, \texttt{[1]}\}, n_{split})$$

Let us assume id(p_n) = [00111101...]. Then p_n knows after the first step some neighbors matching the mask [1], and it knows some neighbors matching [0] (of which some may even be friends of p_n). p_n now asks from a randomly chosen neighbor $p_{[0]}$ matching [0] neighbors for the masks [01], [00]:

$$p_{[0]}\texttt{.getNeighborsOrFriendsForMasks}(\{\texttt{[00]}, \texttt{[01]}\}, n_{split})$$

Then we continue:

$$p_{[00]}\texttt{.getNeighborsOrFriendsForMasks}(\{\texttt{[000]}, \texttt{[001]}\}, n_{split})$$

This will be continued until a call of the form

$$p_{[x]}\texttt{.getNeighborsOrFriendsForMasks}(\{\texttt{[x0]}, \texttt{[x1]}\}, n_{split})$$

reaches a peer $p_{[x]}$ that knows fewer than n_{split} summaries matching [x]. In this case, $p_{[x]}$ will give back the summaries matching [x] as a result. After receiving these summaries, p_n is fully introduced into the Rumorama-network.

Note that some merge operations can follow immediately if the number of the friends found in this way is smaller than $n_{merge}(p_n)$.

In principle a Rumorama peer leaving the network does not need to quit gracefully, because if p_a ceases participation in the rumor spreading process, peers will expire their information about p_a after a time t_{exp}. However, performance can be greatly improved by doing *leave spreading*: within an additional rumor spreading process, peers let other peers know when they failed to contact a given peer. Each peer will now collect "witnesses" for the presence and absence of each other peer within the leaf net. Based on the witnesses' information, peers will be expired from friends lists.

4. SCALABILITY OF RUMORAMA

Within this section we illustrate Rumorama's scalability. In our example, we will estimate the cost and the performance of Rumorama within a very large network. For our estimation, we assume a Rumorama-network consisting of $N = 2^{20} = 1,048,576$ peers. We assume $n_{split} = 50$ and $n_{merge} = 2^4 = 16$. In this case, the maximum friend mask length of a peer within that network will be (with very high probability) $\left\lceil \log \frac{N}{n_{merge}} + \epsilon \right\rceil = \left\lceil \log \frac{2^{20}}{2^4} + \epsilon \right\rceil = 17$. Furthermore, we assume spreading rounds to be 5 minutes of duration.

The calculations are intended to provide a feeling for the behavior of Rumorama. A more formalized treatment of Rumorama with some sketches of proof can be found in [8].

4.1 Estimation of Query Duration

In our setting, a query will be resolved after 17 indirections plus the duration of a PlanetP query processing in the leaf net. If we assume the time unit to be the mean duration of a hop, and if we assume the PlanetP query over 16 to 50 peers to be performed in 5 hop equivalents, this brings us to a query duration of $2 \cdot 17 + 5 = 39$ hops within a very large network. As a comparison: in the same setting, a Chord lookup would take 20 indirections, however, as results do not have to be merged along the multicast tree, the result can be passed back in one hop. So, in a scenario without timeouts, a Rumorama similarity query takes about twice as long as a Chord hashtable lookup. However, similarity queries are inherently more complex than hashtable lookups.

If we assume that at each hop 10% of the peers contacted are offline, we will have to *probe* the peers contacted in order to check if they are online. This (4 messages: probe, probe-return, query, query-return) would drive up query times to $4 \cdot 17 + 5 = 73$ hop times, or in other words, in our model a similarity query would cost 4 times the cost of a Chord DHT lookup.

4.2 Cost for Participating in Rumorama

4.2.1 Neighbor Spreading

In our setting, each Rumorama peer needs to perform neighbor spreading with neighbors matching 17 different masks. One friend address with mask and timestamp is 4 (IP address) +2 (port) +20 (mask) +4 (timestamp) = 30 bytes long. If we assume that the peer asks 8 neighbors for every mask for 10 neighbor addresses and their masks, it will

send $8 \cdot 17 = 136$ `requestRandomNeighborsForMask(.)` messages during every rumor spreading round. As an answer it will receive 1,360 masks and addresses per rumor spreading round ($1,360 \cdot 30 = 40,800$ bytes or $326,400$ bits). If we assume a spreading round duration to be $300s$, this would mean that on average we put $\approx 1,100$ bits per second into neighbor spreading communication, or in other words, 2% of 56K phone modem bandwidth.

4.2.2 Friends Spreading

According to Fig. 3, in our setting a peer with up to 50 friends and 5% churn per round will transfer about 10 peer summaries per round as response to `requestSummaries(.)` messages. If we assume large summaries of $20,000$ bytes, we would transfer $20,000 \cdot 10 \cdot 8 = 1.6$ Mbit as summaries per round. This would lead to about $5,400$ bits per second on average.

If we assume about 1,000 addresses sent as part of friend spreading, each address being 6 bytes (we expect each 20 byte peer ID to be shipped with the peer's summary), this adds another (negligible) 160 bits per second in traffic for friends spreading.

Within this calculation in which we assume very large peer data summaries and a very large network even peers with substandard connectivity can participate in the network. Also note, that there are no central components that are required to have large bandwidth.

4.3 Comparing to Distributed Inverted Files

Now, let us assume a peer participating in a Chord-based distributed inverted file. Assume each peer containing $1,000$ documents. For sake of simplicity, we will assume that each document in one peer contains the same distinct $1,000$ terms. However, the words used differ between peers.

In this case, each peer entering the network has to enter $1,000$ word lists into the inverted file. To this end, it has to find for each word contained in the peer the location of the right inverted list. Then it has to send its new addition to the inverted list. In a network of size 2^{20} locating an inverted list corresponds to sending a (20 bytes) key to $\log 2^{20} = 20$ other peers. Locating $1,000$ lists thus means transferring 400 kbyte simply for *finding* the proper inverted lists. If we assume inverted file entries to be $6 + 4 + 4 = 14$ bytes long (the peer's address and port, document number within the peer, term frequency), the transfer of the addenda to the global inverted lists will be another $14 \cdot 1,000 \cdot 1,000 = 14$ Mbyte or 115.2 Mbit. In comparison, making data known to the network is comprised in Rumorama maintenance messages.

If we assume 5% churn, and if every peer behaved as our model peer, we can calculate the average communication load per peer per 300 seconds. Here we get $0.05 \cdot 115,200,000/300 = 19,200$ bits per second.

Please note that the number of bytes to be shipped for participation in an inverted-file based network is roughly proportional to the number of documents indexed in each peer. So, Rumorama will get better with respect to a distributed inverted file for increasing size of peer data collections and worse for decreasing size. The same, in Rumorama the cost of network maintenance is roughly proportional to the summaries' size.

However, one important point has not been treated in this calculation. A Chord-based inverted file would have to introduce redundancy for the inverted lists. Otherwise peers leaving the network would take out some of the indexing data with them. However, redundant storage of Rumorama summaries is already comprised in the protocol.

4.4 Estimating query cost in Rumorama

When performing queries, the cost of the query will be governed by the total number of peers contacted. When performing the query, in each leaf net at least one peer is contacted. For instance if we assume a mean leaf net size of 25 peers, $40,000$ peers will be contacted in order to reach the leaf nets. In addition peers will be contacted according to the quality of their summaries with respect to the query. Assuming we would contact 5% of the peers in a best-match query setting, this would mean another $50,000$ peers to contact, yielding a communcation load of $90,000 \cdot 30 = 2,700,000$ bytes for shipping a three word (30 byte) query. Depending on the document frequencies of the search term employed, this lies orders of magnitude above or orders of magnitude below the amount of data to be sent for performing such a query in a distributed inverted file environment, as the document frequency determines the size of the inverted lists to be shipped. However, the main difference is here, that the communication load will be distributed evenly over $90,000$ peers in Rumorama, instead of 3 peers being mainly involved in a 3-keyword query performed on a distributed inverted file.

A similar scenario applies for multimedia queries as described in [9]. Here exact queries are prohibitively expensive (they would involve scanning *all* document vectors stored), however approximate similarity queries can be performed using peer data summaries. Due to the summaries not all peers need to be contacted for performing the query, and due to the hierarchical properties of Rumorama, there is load balancing when performing the query.

5. EXPERIMENTS

Rumorama is a framework for similarity search in P2P networks based on summaries. The performance of a Rumorama-based application is thus influenced by the summaries employed with Rumorama. However, the quality of summaries is out of scope within this paper. We point the reader to the broad literature about distributed IR. Starting points could be [1, 4].

Within the experiments described in this section we are interested in Rumorama-specific parameters, notably in the questions if Rumorama scales well and if Rumorama performs well under churn.

The present version of Rumorama was designed for a given usage scenario and tested within a simulation of this usage scenario. The present usage scenario was picked because it is hard for P2P systems. The rationale is that a system being robust to this scenario will also perform in less adverse scenarios. The scenario chosen is inspired by the real world scenario described in [15] and can be described as follows.

5.1 Usage Scenario

We simulated a network in which peers have an exponential lifetime distribution: A peer enters the network, and it has $P = 1 - r_c$ probability of surviving the next round, *i.e.* the next time interval of length t_r. We assume the peer to be gone for good. It won't re-enter the system.

The reason why this model is hard for P2P systems is

Parameter	Default value	Meaning
t_r	300s	Time for a complete rumor spreading round
N	500	Total #peers in the network
(n_{split}, n_{merge})	$(50, 16)$	#peers per leaf net max/min
r_c	0.05 per round (5%)	Fraction of peers to be "churned" in a given time
δ	8	#peers to be contacted peer round
t_{exp}	3,000s	time unused summaries/addresses are kept before expiring them

Table 1: Important parameters and their default values.

twofold: *Firstly,* as every peer has the same leave probability, there is no way to choose something akin to a super-peer, *i.e.* a peer supposed to stay in the network longer than other peers. *Secondly,* the fact that a peer leaves the network for good forces us to expire peers.

The choice of default parameters (see Tab. 1) is consistent with this model. 5% of churn per $t_r = 300s$ means that the half life of a peer is 80 minutes, and that after one day only 10^{-7} peers that started the day are still online. Note that the peers leaving the network in a round of length t_r quit at random points of time during that round.

5.2 Experimental Setup

For our experiments, we implemented Rumorama on a discrete-event simulator of our own making. Within the simulator we modeled the asynchronous remote method invocations of the Rumorama protocol as a packet exchange.

In each experimental run, we simulated 900,000 seconds (*i.e.* 25 hours). We started the run with a network consisting of two peers, increasing the network within 300 (simulated) seconds to size 50, then waiting another 2,700 seconds for the network to stabilize in order to be sure to measure the stable state of the system. Then we measured during 6,000 seconds with the current size of 50 peers, yielding one measurement point on the plot. After the measurement, we increased size by another 50 peers within 300 seconds, waited another 2,700 seconds, measured for 6,000 seconds and so forth.

In the following sections we compare Rumorama with our incarnation of PlanetP, *i.e.* a version of Rumorama in which every peer knows all other peers. In the interest of comparability of Rumorama and PlanetP we used the same rumor spreading and expiry methods for both systems (as described in section 3).

5.3 Experiments

In the experiment depicted in Fig. 3, we show that the number of summaries to be shipped by an average peer increases linearly in PlanetP whereas it is bounded in Rumorama. In these experiments, we varied N from 50 to 450 in steps of 50, for each N as described in the previous section. The values for the other parameters are given in Tab. 1. We also did measurement runs on Rumorama for 500, 750 and 1,000 peers.

Two things are noteworthy about this plot: *Firstly,* the number of summaries shipped is larger than the number of peers actually churned (replaced) per leaf net and time interval of length t_r. This is due to the following effects: it can happen that a peer asks for and receives the same summary from two or more other peers. This can be avoided, but avoiding double sending of summaries would reduce coverage, which is why we opted for the rumor spreading method presented here.

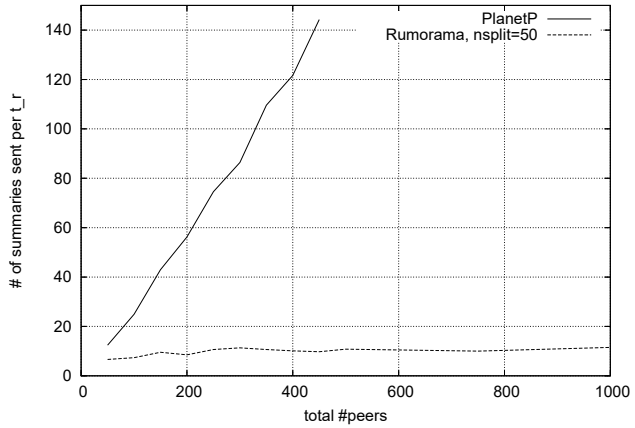

Figure 3: This figure compares the number of summaries to be transmitted by the average PlanetP peer with the number of summaries to be transmitted by the average Rumorama peer. It is clearly visible that in PlanetP the amount of summaries sent per round grows linearly with the size of the network, while the amount of summaries to be sent by a Rumorama peer is bounded.

Secondly, although $n_{split} = 50$ a peer in a Rumorama network with $N = 50$ sends on average fewer summaries per peer than a PlanetP network of size $N = 50$. This is due to churn. Each peer knows in addition to the live peers a small but significant number of peers of which it *believes* that they are alive, but which are in fact offline and not yet expired. As split and merge take into account the number of friends stored instead of the number of friends alive, even a Rumorama network with 50 live peers will consist of multiple smaller leaf nets.

We define *friend coverage* as the number of live friends known to a peer divided by the actual number of live friends. We define as *query coverage* the number of summaries of live peers reached during a query divided by the actual number of live peers during that query. As shown in [8], the loss of coverage has a large impact on result quality. As a consequence, we want to investigate if Rumorama networks manage to provide good coverage in the presence of churn. As Fig. 4 shows, Rumorama does a good job in maintaining good coverage in the presence of varying levels of churn.

Fig. 4 contains two curves, the friends coverage and the query coverage. Here, friends coverage designates the fraction of live peers in its leaf net known to a peer: For example, if there are 10 live friends matching p_a's friend mask,and p_a

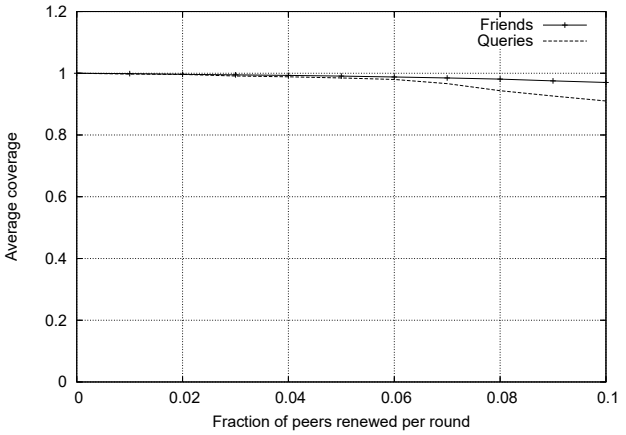

Figure 4: The impact of churn on the coverage. Friend coverage stays remarkably close to 1. Query coverage decreases a bit more strongly, the reason being that summary spreading lags behind friend spreading.

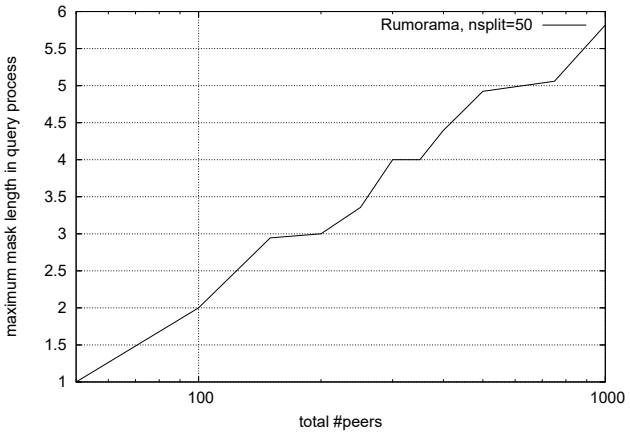

Figure 5: The number of hops per query depending on the size of the network. Note that the x scale is logarithmic. As the number of hops varies logarithmically with the size of the collection, the plot shown is roughly linear.

knows 6 of them, its friends coverage is 60%. Query coverage is the actual coverage measured in query runs.

The figure illustrates the loss of coverage incurred by the separation of friend address spreading and asking for summaries (see section 3.1): The coverage of queries is slightly lower than that of friends. However, the communication savings justify the separation of friends spreading and summary spreading.

We also measured the maximum mask length encountered, *i.e.* the number of hops a query had to traverse. As the number of hops is given by the maximum mask length encountered, and the mask length increases logarithmically with the network size, we would expect the number of hops to increase logarithmically with network size. Fig. 5 shows that this is the case.

We can deduce from the experiments that Rumorama scales well for large P2P networks and achieves a high coverage under churn.

6. CONCLUSION

In this paper we have presented Rumorama, a scalable framework for similarity search in P2P systems based on the dissemination of summaries. The scalability of Rumorama networks is achieved by a combination of rumor spreading for the dissemination of summaries, query processing by source selection based on summaries, and structured multicast of queries to peers such that each peer summary is considered exactly once.

While the present publication shows the scalability and the viability of Rumorama, it also shows opportunities for future work. Possible directions include investigating optimization opportunities for Rumorama for more application scenarios, investigating Rumorama-optimized rumor spreading in the vein of [3, 6], and the efficient, approximate calculation of collection-wide information via rumor spreading.

7. REFERENCES

[1] J. P. Callan, Z. Lu, and W. B. Croft. Searching Distributed Collections with Inference Networks. In *Proc. 18th ACM SIGIR*, Seattle, Washington, 1995.

[2] F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. Technical Report DCS-TR-483, DCS, Rutgers University, 2002.

[3] A. Datta, M. Hauswirth, and K. Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *Proc. of the 23rd Intl. Conf. on Distributed Computing Systems, ICDCS2003*, 2003.

[4] L. Gravano, H. Garcia-Molina, and A. Tomasic. The Effectiveness of GlOSS for the Text Database Discovery Problem. In *SIGMOD Conf.*, Minneapolis, MN, USA, 1994.

[5] R. Huebsch, J. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. In *Proceedings of VLDB 2003*, Berlin, Germany, 2003.

[6] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proc. 41st FOCS*. IEEE Computer Society, 2000.

[7] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, and R. Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In M. F. Kaashoek and I. Stoica, editors, *IPTPS*, volume 2735 of *LNCS*. Springer, 2003.

[8] W. Müller, M. Eisenhardt, and A. Henrich. Scalable Summary Based Retrieval in P2P Networks. Technical report, Bamberg University, 2005.

[9] W. Müller and A. Henrich. Fast Retrieval of High-Dimensional Feature Vectors in P2P Networks Using Compact Peer Data Summaries. In *ACM MIR'03 Workshop*, Berkeley, CA,USA, 2003.

[10] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks. In *In Proc. of the 12th Intl. World Wide Web Conf., Budapest*, 2003.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. Conf. on applications, technologies, architectures, and protocols for computer communications*, San Diego, CA, USA, 2001.

[12] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. ACM SIGCOMM Conf.*, San Diego, CA, USA, 2001.

[13] C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proc. NSDI*, 2004.

[14] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, 2002.

[15] B. Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In F. Kaashoek and A. Rowstron, editors, *1st Intl. Workshop on Peer-to-Peer Systems*, 2002.