CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2006; **?**:1–13 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Proactive gossip-based management of semantic overlay networks

Spyros Voulgaris[1], Maarten van Steen[2], and Konrad Iwanicki[2,*]

[1] *Department of Computer Science*
ETH Zurich
Haldeneggsteig 4, 8092 Zurich, Switzerland
`spyros@inf.ethz.ch`

[2] *Department of Computer Science*
Vrije Universiteit Amsterdam
De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands
{`steen,iwanicki`}`@few.vu.nl`

SUMMARY

Much research on content-based P2P searching for file-sharing applications has focused on exploiting semantic relations between peers to enhance searching. Current methods suggest *reactive* ways to manage semantic relations: they rely on the usage of the underlying search mechanism, and infer semantic relations based on the queries placed and the corresponding replies received. In this paper we introduce a different approach, proposing a *proactive* method to build semantic overlays. Our method is based on an epidemic protocol that clusters peers with similar content. Peer clustering is done in a completely implicit way, that is, without requiring the user to specify any preferences or to characterize the content of files being shared. In our approach, each node maintains a small list of semantically similar peers. Our simulation studies show that such lists are highly effective when searching for content. The construction of these lists through epidemics is efficient and robust, even in the presence of changes in the network.

KEY WORDS: Peer-to-Peer Search Networks, Gossip-Based Protocols, Semantic Overlay Networks

Contract/grant sponsor: *Cooperation DevLab, Development Laboratories, The Netherlands.

1. INTRODUCTION

Locating data among thousands or millions of nodes is clearly of great importance in today's Internet-based information age. This has been broadly demonstrated by the enormous popularity of file sharing peer-to-peer (P2P) systems in recent years. The inherent limitations of centralized systems (e.g., scalability, failure tolerance, and administration costs), as well as the desire to avoid centralized control over content searching (e.g., censorship or result manipulation by political or financial interests), have urged researchers to seek decentralized solutions. This has resulted in a wealth of proposals for content-based searching on peer-to-peer networks [1, 2, 3, 4, 5].

In this paper, we focus on enhancing decentralized search for content by forming *semantic overlay networks*, that is, by forming links among semantically related peers. In such a framework, a node searches for content by first querying its semantically close peers, and then resorting to search methods that span the entire network only if the former attempt provided no satisfactory results.

One way to capture semantic relationships among nodes is through analysis of query results, leading to the construction of a local *semantic list* at each node. Semantic lists consist of references to other, semantically close peers. However, this method implies a "warm up" phase for semantic lists to become useful. Even worse, semantic lists may indefinitely remain in a warm up phase, if node churn is high and search rate low.

In this paper we are interested in solutions where semantic relationships between nodes are captured *proactively*, independently of the execution of queries. When a node performs a search, the appropriate semantic links are already in place, yielding a significant enhancement already from the first query.

A recent study on search methods in structured, unstructured, or hybrid form peer-to-peer networks [6], reveals that virtually all semantic-based peer-to-peer search schemas follow an integrated approach towards the construction of semantic lists, while at the same time accounting for changes occurring in the set of nodes. These changes involve the joining and leaving of nodes, as well as changes in a node's preferences.

Maintaining a semantic overlay network in the face of node churn is a non-trivial task. Such networks exhibit high clustering, which greatly facilitates searching for semantically related content when nothing changes. However, their nature does not favor dynamic scenarios: handling dynamics requires the discovery and propagation of changes that may happen *anywhere* in the network, rather than only in a node's semantic neighborhood. Thus, overlay networks should additionally reflect desirable properties of random graphs and complex networks in general [7, 8]. These two conflicting demands generally lead to complexity when integrating solutions into a single protocol.

Content-based searching frameworks should decouple these two concerns. On one hand, with respect to constructing and using semantic lists, the protocol should optimize these lists for search exclusively, regardless of any other desirable property of the resulting overlay. On the other hand, when it comes to handling network dynamics, a separate protocol should be employed to provide up-to-date information. The combination of the two protocols should allow proper adjustments in the semantic lists, leading to adjustments in the semantic overlay network itself.

In this work we propose such an approach for managing semantic overlay networks based on two layers. The top layer contains a gossip-based protocol that strives to optimize semantic lists for searching only. The bottom layer concentrates on sampling changes that may happen anywhere in the network, in a fully decentralized fashion, similar in nature to the peer-sampling service described in [9]. Our principal contribution is that we demonstrate that this two-layered approach leads to high-quality semantic overlay networks. We substantiate our claims through extensive simulations on static, dynamic, and seriously damaged networks, using traces collected from the eDonkey file-sharing network [10].

The paper is organized as follows. We start with presenting our protocols in the next section, followed by describing our experimental setup in Section 3. Performance evaluation is discussed in Section 4, followed by an analysis of consumed bandwidth in Section 5. We conclude with a discussion in Section 6.

2. THE PROTOCOL

In our model each node maintains a dynamic list of semantic neighbors, called its *semantic view*, of fixed small size ℓ. A node searches for a file by first querying its semantic neighbors. If no results are returned, the node then resorts to the default search mechanism.

2.1. Outline

Our goal is to maximize the hit ratio of the first phase of the search, so as to lower the burden of the—typically significantly more expensive—second phase. We will call this the *semantic hit ratio*. We anticipate that the probability of a neighbor satisfying a peer's query is proportional to the semantic proximity between the peer and its neighbor. Consequently, the semantic hit ratio of a node is expected to be maximized by selecting the ℓ semantically closest peers out of the whole network in its semantic view.

We consider a *semantic proximity function* $S(P, Q)$ which quantifies the semantic proximity between two peers P and Q, based on the lists of files they hold. The semantically closer the file lists of P and Q, the higher the value of $S(P, Q)$. We are essentially aiming at finding peers $Q_1, Q_2, ..., Q_\ell$ for peer P's semantic view, such that the sum $\sum_{i=1}^{\ell} S(P, Q_i)$ is maximized.

Such a semantic proximity function is typically expected to exhibit transitivity to a certain extent: if P and Q are semantically similar to each other, and so are Q and R, then some similarity between P and R is likely to hold. Note that this transitivity does not constitute a hard requirement for our system. In its absence, semantically related neighbors are discovered based on random encounters. If it exists though, it is exploited to dramatically enhance efficiency.

2.2. Design motivation

From our previous discussion, the goal is to build, for each node, a semantic view comprising the node's ℓ closest semantic neighbors out of the whole network. And these lists should

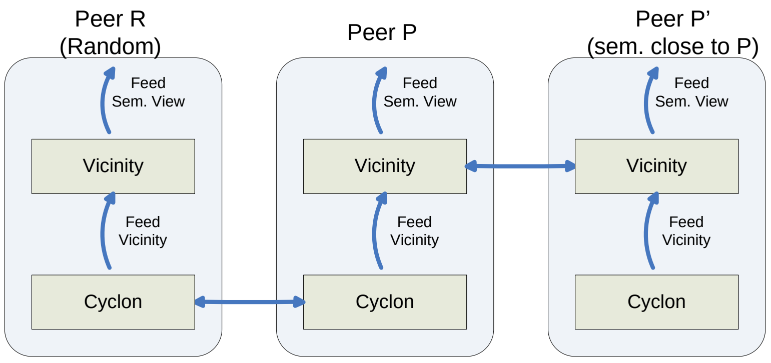

Figure 1. The two-layered framework

be dynamically adapted to reflect the current network state. There are two sides to this construction.

First, exploiting the transitivity property of the semantic proximity function S, a peer should explore nodes that are semantically close to its neighbors. In other words, if Q is in the semantic view of P, and R in the semantic view of Q, it makes sense to check whether R is also semantically close to P. Exploiting the transitivity in S should then quickly lead to high-quality semantic views.

Second, candidates from all over the network should be examined. The problem with examining only neighbors' neighbors, is that we will be eventually searching only in a single semantic cluster, although a good semantic neighbor may have just joined at a random point in the network. Similar to the special "long" links in small-world networks [11], we need to establish links to *other* semantically-related clusters. Likewise, when new nodes join the network, they should easily find an appropriate cluster to join. These issues call for a randomization when selecting nodes to inspect for adding to a semantic view.

Our design decouples these two aspects by adopting a two-layered set of gossip protocols, as shown in Figure 1. The lower layer, called CYCLON [12], is in charge of maintaining a connected overlay and of periodically feeding the top-layer protocol with nodes uniformly randomly selected from the whole network. The top-layer protocol, called VICINITY, focuses on discovering peers that are semantically as close as possible, and of adding these nodes to the semantic views.

2.3. Gossiping framework

Communication between peers is carried out by means of *gossip items*, or simply *items*. A gossip item originated by peer P is a tuple containing the following three fields:

1. Contact information of P (network address and port)
2. Creation time of the item
3. Application-specific data; in this case the file list of P

```
/*** Active thread ***/
// Runs periodically every T time units
q = selectPeer()
myItem = (myAddress, timeNow, myFileList)
buf_send = selectItemsToSend()
send buf_send to q
receive buf_recv from q
view = selectItemsToKeep()

/*** Passive thread ***/
// Runs when contacted by some peer
receive buf_recv from p
myItem = (myAddress, timeNow, myFileList)
buf_send = selectItemsToSend()
send buf_send to p
view = selectItemsToKeep()
```

Figure 2. Epidemic protocol skeleton

Each node locally maintains a number of items per protocol, called the protocol's *view*. This number is the same for all items, and is called the protocol's *view size* (ℓ_v for VICINITY, and ℓ_c for CYCLON).

Figure 2 presents the generic skeleton for both VICINITY and CYCLON gossiping protocols. Each node executes two threads. An *active* one, which periodically wakes up and initiates communication to another peer, and a *passive* one, which responds to the communication initiated by another peer.

The functions appearing underlined, namely `selectPeer()`, `selectItemsToSend()`, and `selectItemsToKeep()` form the three *hooks* of this skeleton. Different protocols can be instantiated from this skeleton by implementing specific policies for these three functions, in turn, leading to different emergent behaviors.

Another parameter, called *gossip length* (g_v for VICINITY, and g_c for CYCLON), controls the number of items exchanged in each communication.

For VICINITY, we chose the policies shown in Figure 3(a). Note that `selectItemsToKeep()` takes into account CYCLON's view too in selecting the best ℓ_v items to keep. This is the default link between the two layers. As we discuss below, COMPLETE will turn out to be an excellent choice for forming semantic clusters.

For CYCLON, we made the choices shown in Figure 3(b). CYCLON is a protocol we previously developed, and which is extensively described and analyzed in [12]. Effectively, what `selectItemsToSend()` and `selectItemsToKeep()` establish is an *exchange* of some neighbors between the views of the two communicating peers. In addition to that, the selected peer's item in the initiator's view is always removed, but the initiator's (new) item is always placed in the selected peer's view.

Hook	**Description**
`selectPeer()`	Select item with the oldest timestamp
`selectItemsToSend()`	
Random	Randomly select g_v items
Selective	Select the g_v items of nodes semantically closest to the selected peer
Complete	Select the g_v items of nodes semantically closest to the selected peer from the Vicinity view *and* the Cyclon view
`selectItemsToKeep()`	Keep the ℓ_v items of nodes that are semantically *closest*, out of items in its current view, items received, and items in the local Cyclon view. In case of multiple items from the same node, keep the one with the most recent timestamp.

(a)

Hook	**Description**
`selectPeer()`	Select item with the oldest timestamp
`selectItemsToSend()`:	
active thread	Select own item and randomly $g_c - 1$ others from the Cyclon view
passive thread	Randomly select g_c items from the Cyclon view
`selectItemsToKeep()`:	
active thread	Keep all g_c received items, replacing (if needed) the item selected by `selectPeer()` and the $g_c - 1$ ones selected to send.
passive thread	Keep all g_c received items, replacing (if needed) the g_c ones selected to send.
	In case of multiple items from the same node, keep the one with the most recent timestamp.

(b)

Figure 3. The chosen policies for (a) the Vicinity protocol and (b) the Cyclon protocol.

Cyclon creates an overlay with completely random, uncorrelated links between nodes, such that the in-degree (the number of incoming links) is practically the same for each node. Importantly, it can achieve this property fairly quickly even when a small number of items (such as 3 or 4) is exchanged in each communication, even for large views of several dozens of items. Therefore, it is ideal as a lightweight service that can offer a node a randomly selected peer from the current set of nodes.

3. EXPERIMENTAL ENVIRONMENT AND SETTINGS

Our experiments were carried out using PeerSim [13], an open source simulator for P2P protocols, developed in Java.

We used real world traces from the eDonkey file sharing system [14], collected by Le Fessant et al. in November 2003 [10]. These traces log a set of 11,872 world-wide distributed peers along with the files each one shares. A total number of 923,000 unique files is being collectively shared by these peers.

In order to simplify the analysis of our system's emergent behavior, we chose equal gossiping periods for both layers. More specifically, once every T time units each node initiates first a gossip exchange with respect to its bottom (CYCLON) layer, immediately followed by a gossip exchange at its top (VICINITY) layer. Note that even though nodes initiate gossiping at universally fixed intervals, they are not synchronized with each other.

Although the protocols of both layers are asynchronous, it is convenient to introduce the notion of *cycles* in order to study their evolutionary behavior with respect to time. We define a cycle to be the time period during which *each* node has initiated gossiping exactly *once*. Since each node initiates gossiping periodically, once every T time units, a cycle is equal to T time units.

In the remaining of this section, we list the parameters of our system, and their respective values.

Proximity Function S We defined the proximity S between two nodes P and Q, holding file lists F_P and F_Q respectively, to be the number of files that lay in both lists. More formally: $S(P,Q) = |F_P \bigcap F_Q|$. As stated in 2.1, the semantically closer two nodes are, the higher the value of S is. Note that our goal was to demonstrate the power of our gossiping protocol in forming semantic overlays based on a given proximity function. Although much richer proximity functions could have been applied, it was out of the scope of this paper.

Semantic view size ℓ We arbitrarily defined the semantic view to consist of the 10 semantically closest peers in the VICINITY view. As shown in [5], a semantic view size of $\ell = 10$ provides a good tradeoff between the number of nodes contacted in the semantic search phase and the anticipated semantic hit ratio.

Gossip view size ℓ_v **and** ℓ_c With respect to the view size selection, we are faced with the following tradeoff for both protocols. Clearly, a large view gives the protocol more choices in selecting semantically closer neighbors, and therefore accelerates the construction of (near-)optimal semantic views. On the other hand, in the presence of a dynamic environment, a large view is harder to maintain: the larger the view, the longer it takes to contact all peers in it, resulting in the accumulation of stale links to disconnected nodes. Of course, larger views also take up more memory, although this does not constitute a substantial constraint for modern computers.

Considering this tradeoff, and based on experiments not further described here, we fixed the view size to 100 as a basis to compare different configurations. When both VICINITY and CYCLON are used, they are allocated 50 view entries each.

Gossip length g_v **and** g_c The gossip length, that is, the number of items sent per gossip exchange per protocol, has a direct influence on the amount of bandwidth used. This becomes of greater importance, given that an item carries the file list of its respective node. So, even though exchanging more items per gossip exchange allows information to

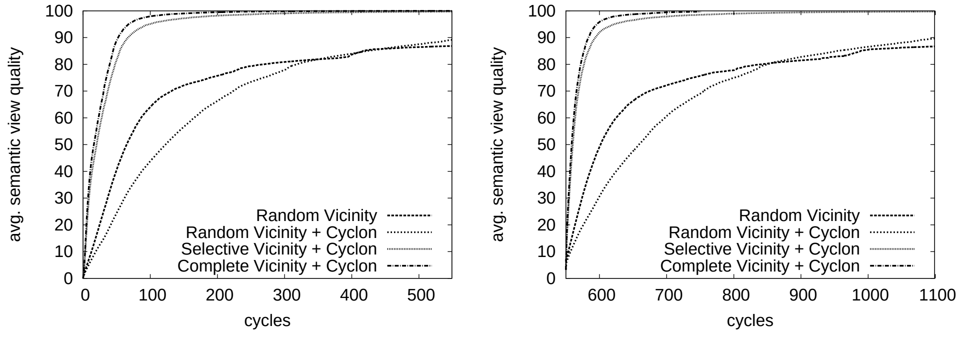

Figure 4. (a) Convergence of sem. views' quality. (b) Evolution of semantic views' quality for a sudden change in all users' interests at cycle 550.

disseminate faster, we are inclined to keep the gossip lengths as low as possible, as long as the system's performance is reasonable.

For the sake of a fair comparison, we fixed the total gossip length to a low value of 6 items. When both Vicinity and Cyclon are used, each one is assigned a gossip length of 3.

Gossip period T The gossip period does *not* have any effect on the protocol's behavior. The overlay evolves as a function of the number of messages exchanged by the two layers, or, equivalently, of the number of cycles elapsed. The gossip period only affects how fast this evolution will take place in real time. The only constraint is that the gossip period T should be adequately longer than the expected latency throughout the network, so that gossip exchanges are not favored or hindered due to latency heterogeneity. A typical gossip period for our protocol would be one minute, even though this does not affect the following analysis.

4. PERFORMANCE EVALUATION

4.1. Convergence speed on cold start

The proximity function dictates that in a converged state, each node has discovered the ℓ peers that share the largest number of common files with it. We were interested in how fast our algorithm drives the system towards that state, when starting with a random, sparsely connected network. This is a crucial property as it determines the time necessary to bootstrap the network or to recover from (massive) failures.

As the metric for evaluating the convergence speed of our algorithm, we used a node's *semantic view quality*, defined as the ratio of the number of common files a node shares with its current ℓ semantic neighbors, over the number of common files it would share with its ℓ optimal semantic neighbors.

Figure 4(a) presents the average semantic view quality as a function of the cycle for four distinct experimental configurations of the protocols. At the beginning of every experiment, each node knew only 5 other random nodes, simply to ensure initial connectivity in a single connected cluster. In favor of comparison fairness, the view size and gossip length were 50 and 3, respectively, in each layer, for all configurations. The only exception was the first configuration, which had a single layer. In that case, the view size and gossip length were 100 and 6, respectively.

In the first configuration, Random Vicinity is the only protocol running. The improvement of the semantic views' quality is rather steep in the first 100 cycles, but as nodes gradually concentrate on their very own neighborhood, the probability of discovering new, possibly better peers decreases and thus, the improvement slows down, eventually stabilizing to a non-optimal state. At that point, although most optimal semantic links have been established, semantically close nodes that happen to be connected in disjoint clusters have no chance of discovering each other. Even worse, in the presence of node churn, new nodes have little or no chance of navigating to their appropriate clusters.

The second, two-layered configuration consists of Random Vicinity and Cyclon running in parallel. The slow start compared to stand-alone Vicinity is a reflection of the smaller Vicinity view (3 as opposed to 6). However, the advantage of the two-layered approach becomes apparent later, when Cyclon keeps feeding the Random Vicinity layer with new, uniformly randomly selected nodes, maintaining a higher improvement rate, and outperforming stand-alone Vicinity in the long run. Note that, due to the continuous feed of random nodes to the Vicinity layer, the semantic links of each node will eventually converge to the optimal ones. Likewise, in a dynamic scenario, newly joined nodes will be able to eventually locate their appropriate clusters. The following two Vicinity optimizations focus on speeding up this convergence.

The third configuration, demonstrates the benefits of using Selective Vicinity, which significantly improves the initial phase of building a semantic overlay network. Such a behavior is to be expected, since the items sent over in each Selective Vicinity communication, are the ones that have been selected as the semantically closest to the recipient.

Finally, in the fourth configuration, Complete Vicinity keeps the improvement rate high even when the semantic views are very close to their optimal state. This is due to the broad random sampling inherent to this configuration. In each communication, a node is offered the semantically closest peers selected not only among its neighbor's 50 semantic links, but *also* among its neighbor's 50 random links. In this way, semantically related peers that belong to separate semantic clusters quickly discover each other, and subsequently such two clusters merge into a single cluster virtually in no time.

4.2. Adaptivity to changes in user interests

Users' interests often evolve over time, and therefore, the semantic overlay must adapt accordingly. To test our protocol's adaptivity to user interest dynamics, we ran experiments where the interests of some users changed. We simulated the interest change by picking a random pair of nodes and swapping their file lists in the middle of the experiment. At that point, two such nodes found themselves with semantic views unrelated to their (new) file lists. Consequently, they had to gradually rebuild the semantic views by replacing the useless links with new ones, pointing to the semantically closest nodes.

To detect any possible flaws in our protocol, we employed the worst case—practically unrealistic—scenario: *all* nodes changed interests at once, at cycle 550 of the experiment in Figure 4(a). The evolution of the average semantic view quality from the moment when all nodes changed their interests, is presented in Figure 4(b). We see that not only does the system converge as in the previous experiment, but also, the convergence is faster compared to Figure 4(a). Such a behavior is due to the fact that at cycle 550 views are already fully filled up, so nodes have more choices when they start looking for good candidate neighbors. Consequently, although the above scenario is very unrealistic, it demonstrates the power of our protocol in adapting to even massive-scale changes. This adaptiveness is due to the priority given to newer items in `selectItemsToKeep()`, which allows a node's items with updated semantic information to replace older items of that node relatively fast.

4.3. Effect on semantic view hit ratio

Even though this paper does not propose a fully-fledged search system, we claim that a semantic overlay, as constructed by our protocol, endorses P2P searching. More specifically, we advocate that the semantic links formed to reflect semantic proximity based on current files are useful to locating future requests for files, according to the locality of interest principle.

In order to substantiate this claim we conducted the following simple experiment. A randomly selected file was removed from *each* node, and the system was let run to form a semantic overlay based on the remaining files. Then, we tested the system's efficiency in satisfying future requests by each node performing a search on its missing file, and we recorded the semantic hit ratio of every search.

We measured the semantic hit ratio to be over 36% for a semantic view of size 10. In other words, in more than one out of three searches on average, a node was able to locate a file it is interested in by merely asking its semantic neighbors, without resorting to the—typically expensive—underlying search mechanism.

Figure 5 presents the semantic view hit ratio as a function of the cycle. Three experiments are shown, with gossip lengths for *both* layers set to 1, 3, and 5. Again, the speed of convergence to the optimal value is very high. Note that computation of the hit ratio for each cycle was made off-line, without affecting the mainstream experiment's state.

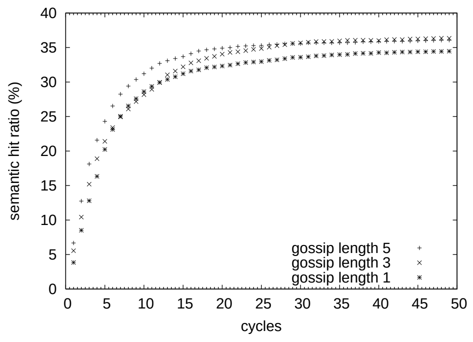

Figure 5. Semantic view hit ratio, for gossip lengths 1, 3, and 5 in each layer.

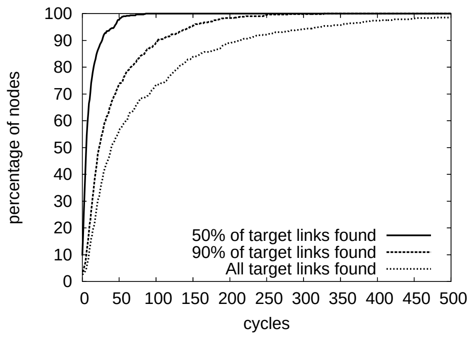

Figure 6. CDF of the speed by which the semantic view of a joining node is filled with optimal neighbors.

4.4. Behavior under changing membership

To investigate the behavior of our algorithm as nodes join and leave the network, we conducted two different experiments with Complete Vicinity. First, we are interested to see how fast a node discovers optimal neighbors for its semantic view, when joining an already converged network.

To this end, we conducted a series of experiments, each time starting with a network from which one randomly selected node was removed. After the overlay had converged, we added the missing node and measured the number of cycles it took to fill, respectively 50%, 90%, and 100% of its semantic view with *optimal* neighbors. The respective cumulative distribution function graphs are shown in Figure 6.

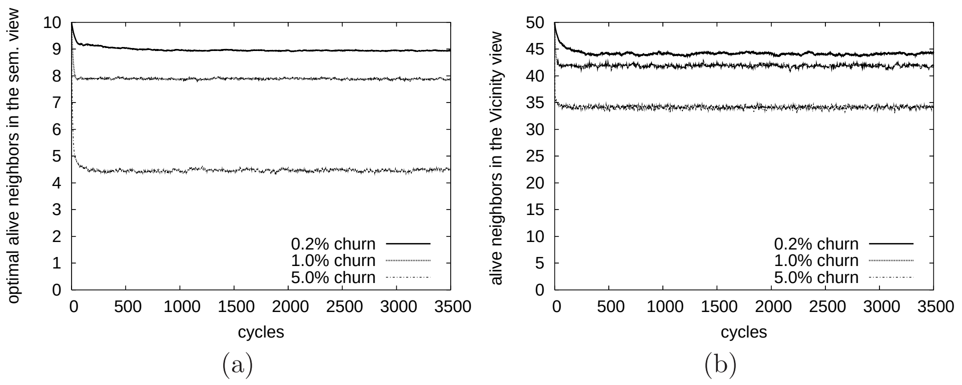

Figure 7. (a) The average number of optimal alive neighbors in the semantic view. (b) The average number of alive neighbors in the VICINITY view.

The experiments clearly show that the semantic view is rapidly filled with optimal neighbors for the vast majority of the nodes, although some may take considerably more time. It can also be seen that, although discovering *all* best neighbors may take arguably long for some nodes, it takes significantly fewer cycles to discover *most* best neighbors (CDFs for finding 50% and 90% of best neighbors shown).

Let us now take a look at how the algorithm behaves under churn, that is, when nodes regularly join and leave the network. For this experiment, we considered an initial converged network of 10,000 nodes. During each cycle we removed n nodes and replaced them by n other nodes. Every node in the system corresponded to one node from the eDonkey traces (i.e., stored the same files), which contained a total of 11,872 nodes. Therefore, at any moment in time a random subset of 10,000 out of 11,872 nodes were active, and the remaining 1,872 nodes were down.

The analysis of churn rates from real-world Gnutella traces [15] shows that the set of active nodes changes by approximately 0.2% every 10 seconds. In other words, if we assume a cycle length of 10 seconds, a realistic value for n is 20. We have also experimented with larger churn rates, namely 1% and 5%, which correspond to a cycle duration of 50 and 250 seconds, respectively.

The results of these experiments are shown in Figure 7(a) in which the average number of optimal alive neighbors in semantic views under different churn values is plotted. We see that as the churn rate increases, the semantic views generally remain polluted with links to non-optimal neighbors. This can be easily explained by considering the VICINITY view (from which the neighbors for the semantic views are extracted). In Figure 7(b), we see that under high churn the view contains a relatively large fraction of links to dead nodes. As these inactive links may refer to nodes with a large number of common files, they prevent establishing optimal links with nodes that are alive but share a smaller number of files. Moreover, when a node is reborn, it can take some time for other nodes, which now may have non-optimal semantic views, to establish links to it (cf. Figure 6).

5. BANDWIDTH CONSIDERATIONS

The periodic nature of gossiping may potentially inhibit a high usage of network resources (i.e., bandwidth). Below, we investigate the bandwidth price to pay for rapidly converging, high-quality semantic overlays, as constructed by our protocol.

A node gossips on average twice in every cycle (exactly once as an initiator, and on average once as a responder). Each gossip involves $2 \cdot (g_v + g_c)$ items being transferred to and from the node, resulting in a total traffic of $4 \cdot (g_v + g_c)$ items for a node per cycle. The size of an item is dominated by the file list it carries. In P2P file-sharing applications, a file is often identified using a 128-bit (16-byte) value of a one-way hashing function (e.g., MD4 or SHA-1). An analysis of the eDonkey traces [10] revealed an average number of 100 files per node (more accurately, 99.35). Consequently, the file list of a node consumes 1,600 bytes on average. So, in each cycle, the total number of bytes transferred *to* and *from* the node is $6,400 \cdot (g_v + g_c)$. With the gossip length $g_v = g_c = 3$, this amounts to roughly 38,400 bytes, whereas for the gossip length $g_v = g_c = 1$, it is just 12,800.

Maintenance of almost optimal semantic views requires frequent gossiping to account for the node churn. Based on the traces, to achieve 90% optimality of the semantic views, the churn rate must be limited to approximately 0.2%. Consequently, the gossip period T must be equal to 10 seconds, which translates to an average bandwidth of 3,840 bytes per second for $g_v = g_c = 3$. However, if 80% optimality is acceptable, the gossip period can be reduced to 50 seconds, which yields 768 per second, nearly a factor of 5 improvement traded for only 10% quality degradation.

In our view, such a bandwidth consumption is rather small, if not negligible compared to the bandwidth used for the actual file downloads and search requests. We believe that it is, in fact, a small price to pay for relieving the default search mechanism from about 36% of the search load, which is often significantly higher (e.g., flooding or random-walk search). Moreover, the bandwidth can be further reduced by employing techniques such as Bloom filters [16] and on-demand fetching of the file lists, instead of associating a full file list with each view item. Finally, we may dynamically control the gossip period T based on the stability of the system, using techniques similar to [17].

6. DISCUSSION AND RELATED WORK

To the best of our knowledge, all earlier algorithms for implicit building of semantic overlays operate in a *reactive* manner [1, 4, 5]. More specifically, they rely on heuristics to decide which of the peers that served a node recently are likely to be useful again for future queries. One of the hidden assumptions those algorithms make is that the network is *static* (i.e., free of node departures). However, the traces obtained from several real-world systems provide evidence that the highly dynamic nature of file-sharing communities contradicts this assumption.

Similarly, it is not clear how reactive algorithms perform in the presence of dynamic user preferences.

Moreover, as opposed to the existing solutions, our algorithm can, to some extent, help against so-called free-riders in the P2P file sharing networks [18]. Free-riders provide no files to be downloaded by other users, but still use the network to obtain files that are interesting for themselves. Due to the lack of shared files, VICINITY does not create any meaningful semantic views for such misbehaving nodes. This, combined with heuristics forbidding frequent usage of the backup search algorithms, minimizes the number of successful searches for free-riders, and consequently discourages the free-riding practice.

Considering decentralized P2P clustering based on proximity, our work comes close to the T-Man protocol [19], which has been developed independently. Both the protocols, given an appropriate proximity function, can construct various network topologies [20], like grids, toruses, Chord lattices, etc. Although there were significant differences with the original T-Man protocol, the most recent version shows a strong similarity with our work. An important difference remaining is that VICINITY and CYCLON offer the means for controlling bandwidth by arbitrarily deciding on the number of entries that need to be exchanged. This is important in complex, layered, real-world systems, in which efficient bandwidth provisioning may be challenging. More important, however, is that we show in this paper how our specific two-layered gossiping approach can be successfully applied to facilitate searching in P2P file-sharing systems. To the best of our knowledge, such an evaluation has not yet been done before.

To sum up, in this paper, we argued that epidemics can be employed to proactively build and maintain semantic overlays in a large-scale, dynamic file-sharing system. In particular, we advocated using a two-layered approach combining two epidemic protocols: a protocol for clustering semantically related peers and another protocol providing a stream of random peers from the whole system. We showed through simulations using real-world traces how this multi-layer approach ensures rapid convergence of semantic overlays, and at the same time, guarantees high adaptivity and low overhead.

ACKNOWLEDGEMENT

We would like to thank Fabrice Le Fessant for providing us with the eDonkey2000 traces[10] he gathered in November 2003.

REFERENCES

1. K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *INFOCOM Conference*, 2003.
2. Rachid Guerraoui, Sidath B. Handurukande, Kevin Huguenin, Anne-Marie Kermarrec, Fabrice Le Fessant, and Etienne Riviere. Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. In *Peer-to-Peer Computing*, pages 12–22, 2006.
3. Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. P2p content search: Give the web back to the people. In *IPTPS*, 2006.
4. S. Voulgaris, A. Kermarrec, L. Massoulié, and M. van Steen. Exploiting semantic proximity in peer-to-peer content searching. In *10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004)*, Suzhu, China, November 2001.

5. S. Handurukande, A.-M. Kermarrec, F. Le Fessant, and L. Massoulié. Exploiting semantic clustering in the edonkey p2p network. In *11th ACM SIGOPS European Workshop (SIGOPS)*, Leuven, Belgium, September 2004.
6. John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: search methods. *Computer Networks*, 50(17):3485–3521, 2006.
7. Reka Albert and Albert-Laszlo Barabasi. Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, 74(1):47–97, January 2001.
8. M.E.J. Newman. Random Graphs as Models of Networks. In S. Bornholdt and H. G. Schuster, editors, *Handbook of Graphs and Networks: From the Genome to the Internet*, chapter 2. John Wiley, New York, NY, 2002.
9. M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In *Middleware 2004*, volume 3231 of *Lect. Notes Comp. Sc.*, Berlin, October 2004. ACM/IFIP/USENIX, Springer-Verlag.
10. Fabrice Le Fessant, S. Handurukande, Anne-Marie Kermarrec, and Laurent Massoulié. Clustering in peer-to-peer file sharing workloads. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, USA, February 2004.
11. Duncan J. Watts. *Small Worlds, The Dynamics of Networks between Order and Randomness.* Princeton University Press, Princeton, NJ, 1999.
12. Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.
13. PeerSim. `http://peersim.sourceforge.net/`.
14. eDonkey. `http://www.edonkey2000.com/`.
15. Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *ACM Multimedia Syst.*, 9(2):170–184, 2003.
16. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422–426, 1970.
17. K. Iwanicki, M. van Steen, and S. Voulgaris. Gossip-based clock synchronization for large decentralized systems. In *Proceedings of the Second IEEE International Workshop on Self-Managed Networks, Systems & Services (SelfMan 2006)*, pages 28–42, Dublin, Ireland, June 2006.
18. Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, 5(10): , October 2000. Avaiable at: `http://firstmonday.org/issues/issue5_10/adar/`.
19. Márk Jelasity and Ozalp Babaoglu. T-Man: Gossip-based Overlay Topology Management. In *Proceedings of Engineering Self-Organising Applications (ESOA'05)*, July 2005.
20. S. Voulgaris. *Epidemic-based Self-organization in Peer-to-Peer Systems.* PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, October 2006.