Build One, Get One Free: Leveraging the Coexistence of Multiple P2P Overlay Networks

Conference Paper *in* Proceedings - International Conference on Distributed Computing Systems · July 2007

DOI: 10.1109/ICDCS.2007.88 · Source: IEEE Xplore

3 authors, including:

Marin Bertier
Institut National des Sciences Appliquées de …

56 PUBLICATIONS **922** CITATIONS

SEE PROFILE

Build One, Get One Free: Leveraging the Coexistence of Multiple P2P Overlay Networks

Balasubramaneyam Maniymaran,
Dept. of Electrical and Computer Eng.,
McGill Universiy,
Montreal, QC, Canada.
bmaniy@cs.mcgill.ca

Marin Bertier,
IRISA / INSA,
Rennes, France.
mbertier@irisa.fr

Anne-Marie Kermarrec
IRISA / INRIA
Rennes, France.
akermarr@irisa.fr

Abstract

Many different P2P overlay networks providing various functionalities, targeting specific applications, have been proposed in the past five years. It is now reasonable to consider that multiple overlays may be deployed over a large set of nodes so that the most appropriate overlay might be chosen depending on the application. A physical peer may then host several instances of logical peers belonging to different overlay networks. In this paper, we show that the coexistence of a structured P2P overlay and an unstructured one may be leveraged so that, by building one, the other is automatically constructed as well[1]*. More specifically, we show that the randomness provided by an unstructured gossip-based overlay can be used to build the routing table of a structured P2P overlay and the randomness in the numerical proximity links in the structured networks provides the random peer sampling required by gossip-based unstructured overlays. In this paper, we show that maintaining the leaf set of Pastry and the proximity links of an unstructured overlay is enough to build the complete overlays. Simulation results comparing our approach with both a Pastry-like system and a gossip-based unstructured overlay show that we significantly reduce the overlay maintenance overhead without sacrificing the performance.*

1 Introduction

Peer-to-peer (P2P) networks have become a dominant part of the Internet traffic due to the tremendous success of file-sharing systems like Bittorrent [2] and KaZaA [4]. It is mainly due to their self-scaling characteristic where each node acts both as a client and server ensuring that the number of servers increases with the size of a system thus avoiding server bottlenecks. Self-scaling is an essential characteristic for an Internet-scale system and, therefore, there have been many efforts in the research and business communities to expand the usage of the P2P systems beyond file-sharing applications. As a result, P2P systems for resource discovery [18, 19], distributed computing [15], and distributed publish-subscribe system [10] have been developed in the recent past.

A P2P system organizes the participating resources in an application-level *overlay* structure of which varies with time depending on the availability of the resources or the nature of the application. P2P overlays can be classified into two main types: *structured* and *unstructured*. Structured overlays [18, 19] tag the peers with node identifiers (node-IDs) thus providing an efficient support for *distributed hash table* (DHT) functionality. Messages are routed from source to destination in a logarithmic (in the size of the system) number of steps. Structured overlays are efficient in discovering uniquely identified resources. At the other end of the spectrum, unstructured overlays do not follow any specific virtual topology, but let the peers arrange themselves in a connected graph. In such networks, discovery messages have to be disseminated using flooding, gossiping, or random walk. Unstructured networks are becoming more and more popular as they are flexible enough to be optimized for specific applications [22, 20].

Structured overlays provide an efficient exact match interface, but they are not as efficient when it comes to performing a *range query* (e.g. query for video files with size "larger than 700MB") or a *keyword search* (e.g. search for all the songs of "Sarah Brightman"). In addition, the maintenance cost of a structured overlay can be high in dynamic environments where the peers leave and join the system frequently. On the other hand, unstructured networks handle range queries and keyword searches more easily and are highly adaptive to dynamic environments. However, unstructured networks are prone to generate many messages

[1] To be completely fair, we build part of each overlay to obtain the full overlays.

27th International Conference on Distributed Computing Systems (ICDCS'07)
0-7695-2837-3/07 $20.00 © 2007 IEEE

for each discovery request and do not always guarantee an exhaustive search. It is actually easy to demonstrate that depending on the application, one or the other type of overlay is clearly more adapted, justifying the deployment of multiple overlays. In this paper, we show how we can leverage such coexistence.

Motivation: As the number of P2P applications increases, it is likely that several P2P overlay networks cohabit on a network so that the best overlay could be chosen depending on the the application. We argue that this cohabitation should be leveraged to create a *joint overlay* in which the overlays share information to reduce the maintenance cost while keeping the same level of performance. Considering the fact that the P2P traffic now dominates the majority of the Internet traffic, it is crucial to study the feasibility of such joint overlays. Further, well known P2P file sharing system clients such as eMule [3] or Ares [1] actually depend on both structured and unstructured networks for various functionalities.

In this paper, we focus on the creation of a joint overlay with a structured and an unstructured overlays. Although this approach can be generalized (Section 6), we consider two specific P2P overlays for this study: the Pastry [9] structured overlay and an *interest*-based unstructured overlay using gossip protocols [22]. Pastry is a structured overlay network, providing DHT functionality. Each Pastry node maintains two sets of data: (1) a leaf set consisting of numerically closest node-IDs and (2) a routing table filled with systematically distributed node-IDs. A interest-based unstructured network is composed of two protocols: (1) a *clustering service* maintaining interest-based links and (2) a *peer sampling service* (RPS) maintaining a set of pointers to random nodes. While the interest-based links connects to close nodes according to a given proximity measure, the random links ensure connectivity and support for handling dynamism.

Contributions: In this paper, we investigate the extent to which the coexistence of several overlays on the same peer can be leveraged. We identify that the overlays have a `primary` and a `secondary` components. The primary components have strong system-constraints while the secondary components do not. In Pastry, the leaf set, consisting of numerically close node-IDs, is the primary component and the routing table is a secondary component. In the interest-based overlays, the primary component is the clustering service tailored for a given application, while the peer sampling becomes the secondary component.

W claim that by maintaining the primary components of two overlays, their secondary ones can be built for free. In the our case, the leaf set of Pastry provides similar random samples that are provided by the RPS of the interest-based overlay. Conversely, the clustering service of in the interest-based overlay provides the potential candidates to fill the Pastry routing tables. This mutualisation keeps the performances of the individual overlays comparable to their original forms while reducing the maintenance overhead by many folds. Our simulations show this trade off.

Roadmap: Section 2 provides brief descriptions of Pastry and interest-based overlays. Section 3 describes our approach to merge these two schemes. Simulation results are presented in Section 4. Section 5 describes some related work. We discuss how our work can be generalized in Section 6 before concluding in Section 7.

2 Background

In this section, we describe the two systems we consider to form a joint overlay: (1) Pastry is a structured overlay and (2) *C-gossip* is a clustered version of a gossip-based unstructured network.

2.1 Pastry

Pastry is a structured P2P overlay providing DHT functionality. Pastry assigns each node a unique *node-ID* chosen uniformly at random. These node-IDs are r digit numbers of base 2^b. Values for r and b are chosen such that the node-IDs are unique with high probability. The node-ID space is considered to be cyclic, that is, the IDs 0 and $(2^{rb}-1)$ are considered to be adjacent.

Each Pastry node maintains two sets of routing entries namely a *leaf set* and a *routing table*. A routing entry maps a node-ID of a node to its IP address. The leaf set contains a fixed number of routing entries whose node-IDs are numerically closer to the local node-ID (*short links*). A routing table contains entries with distant node-IDs (*long links*) and, therefore, provides routing short-cuts. Table 1 provides an example. The Pastry routing tables are r rows tall and 2^b columns wide. The routing entry in the cell $[i, j]$ has the first $i-1$ digits of its node-ID same as the local node-ID and the i^{th} digit as j. There can be empty cells in a routing table just when no node in the system exists, suitable for that cell. Note that, due to the structure of the routing tables as mentioned above, the probability of finding a node to fill a cell exponentially decreases from top row to bottom. Pastry uses long links for routing and provides logarithmic (in terms of system size) routing complexity. Short links are used in the final phases to ensure convergence.

When a new node joins the Pastry overlay, it uses the routing protocol to find the closest node to its own ID and inserts itself adjacent to that node in the overlay. The nodes in the path of the join message contact the joining node to initialize *gossiping* in which they exchange relevant routing

0	1	2	3
02100	12333		32201
20232	21312	22320	
–	23120	23201	
23300		–	–
	23311	–	23313

Table 1. An example routing table of a Pastry node with node-ID 23310 ($r = 5, b = 2$).

table entries to update each others routing tables. This gossiping operations bootstrap the new node and ensure that the information about the new node is disseminated in the system.

The length of the route path is logarithmic only in the node-ID space, not necessarily in terms of the network latency. The metric `stretch` is defined as the ratio of network latency observed in a Pastry route and the latency of the shortest path between the two end nodes. The latest version of the Pastry protocol described in [9] implements *proximity neighbor selection* (PNS) to reduce the stretch factor in Pastry overlays. In PNS, a Pastry node fills its routing table with nodes that are closer in terms of network latency at the same time observing the Pastry routing table rules. PNS is implemented using a gossip-like protocol, where a Pastry node periodically chooses a node from its routing table from a random row and column and exchanges routing table entries belonging to the specific row. The exchanged entries are analyzed and entries pointing to closer nodes are retained. This gossip mechanism also helps to check whether the routing entries are still active, and, therefore, acts as a routing table maintenance mechanism in dynamic environments. However, PNS significantly increases the message overhead of the Pastry protocol. We consider this latest version of Pastry in this paper.

2.2 Interest-based Gossip Protocol

Gossip-based protocols have recently generated a lot of interest from the research community to build unstructured overlay networks. In such protocols each node periodically exchanges information with another node randomly chosen from its *view* (neighborhood set) so that the information is spread in the system in an epidemic manner. Gossip-based algorithms have been used to create random-like topologies in the context of a resource discovery application, to disseminate news snippets in a news casting application and to disseminate neighborhood information in an overlay maintenance application. These schemes differ in the way they choose the peers to communicate with and the way the exchanged information is processed.

The gossip protocol can also be applied to create node clusters in application-specific overlays using different proximity measures (geographic, semantic). Recently *interest-based overlays* have been considered to improve search in P2P file sharing applications. Peers maintain links to nodes having similar interests in the application domain. For example, in a file sharing application, if the *file caches* of a pair of nodes show a large overlap, the nodes benefit from being neighbors. It is been shown that the probability of getting a hit for a required file in the neighborhood is high in such an interest based overlay [12]. This improves the response time while reducing the gossip messages.

In this paper, we focus on a protocol called C-gossip (clustered-gossip) similar to the one proposed in [22] consisting of two layers (Figure 1(a)): a clustering layer and a *random peer sampling* (RPS) layer [13]. These layers run two separate gossip protocols maintaining two separate views. At each gossip round, the gossip refines a node's clustering view to include closer nodes according to a given proximity measure, while the RPS gossip updates its view such that the nodes in the RPS view is a random sample of the global set of nodes. The clustering layer effectively creates clusters by updating its views selectively. This selective clustering may lead to overlay partitions and failure in adapting to dynamic situations. To avoid this, the clustering protocol occasionally chooses nodes from the RPS view to gossip with. This provides the clustering layer with the ability to gossip "out-of-interest" and thus to discover new nodes and avoid overlay partitions.

Gossip protocols produce a constant $O(N)$ message overhead at each gossip period, as each node contacts another one in every period. Even though the complexity is linear, the absolute value of this overhead can be high.

3 Architecture of the Joint Overlay

This section describes the design steps for constructing a joint-overlay.

3.1 Composition of P2P Overlays

We identified that each P2P system is composed of two components: the *primary* component is created with strong constraints from the system protocol and the *secondary* component is no constraints. Consequently, the primary component becomes the fundamental component of the system and has to be strongly maintained, while the secondary component sustains less aggressive maintainance.

The leaf set of the Pastry overlay is identified as its primary component. The routing entries in the leaf set are constrained to have numerically close node-IDs. The leaf set alone is enough to provide connectivity in the overlay and to ensure routing convergence. On the other hand, the routing table is the secondary component and is weakly constrained although the degree of freedom is dependent on the levels in the routing table (the lower the level of the routing table,

the more constrained). Routing table in Pastry is required to achieve a logarithmic routing complexity, but having few cells empty in the table does not affect significantly the performance (in practice there are always some empty cells). Therefore, the Pastry protocol aggressively maintains the leaf sets making sure that they are always updated with the latest system information, while routing table is opportunistically maintained.

The primary component in the C-Gossip (although not the one that ensures connectivity) is the cluster view. The nodes in the cluster view are constrained to be within the nodes semantic proximity. On the other hand, the nodes in the RPS view have no constraints at all and are chosen purely random. The cluster view has to be strongly maintained to ensure reachability of all neighbors, while the RPS view can be very lazily updated. However, note that, the RPS is crucial for ensuring the connectivity.

3.2 Leveraging the Coexistence of Overlays

The main idea behind the joint overlay is to pay (in terms of message overhead) for the maintenance of the two primary components as in the original protocols and get the two secondary ones for free.

The key point is that these two overlays are constructed in two different domains: one in a node-ID space and the other in an application semantic space ('interest' here). Therefore, a set of nodes clustered according to a metric in one domain produces a random set of nodes in the other domain. The leaf set of the Pastry protocol brings together nodes according to the node-ID space, whom are expected to be a random set in the semantic space. Similarly, the nodes clustered according to interest in the clustering view are expected to have a random node-ID distribution. Therefore, a node may use the leaf set of the Pastry protocol to implement the RPS view in the C-gossip, while the nodes from the cluster view may be used to fill up the Pastry routing tables. This is illustrated on Figure 1.

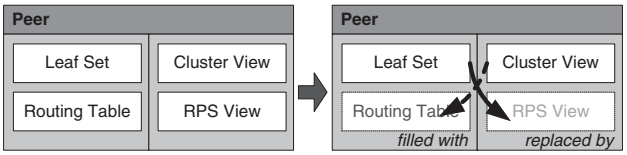


Figure 1. Leveraging cohabiting overlays.

This combination relieves the system of the maintenance of two gossip protocols: the PNS gossip in Pastry and the RPS gossip in C-gossip. Consequently, we expect the message complexity to become almost $1/3$ of the original system implementing two independent overlays.

However, the performances might be affected. As mentioned in Section 2.1, the probability of filling a cell in the Pastry routing table exponentially decreases with the level (from top to bottom). This probability is expected to drop further when the fixed number of nodes from the cluster view is used to fill the routing tables. This issue is reduced by using, not only the local cluster view, but also the views of other nodes encountered in the gossiping protocol. Further, the clustering gossip protocol is expected to converge in the absence of dynamicity. This means that nodes belonging to a given cluster will not change over time in a stable system. Therefore, cluster views may fail to provide enough randomness to fill the routing table once the system become stable. A similar drawback can be observed in using Pastry leaf set to construct the RPS view. The Pastry leaf set also becomes static, once the system stabilizes, leading to the similar performance degradation.

Generally speaking, maintaining overlay networks in a dynamic system is a hard issue. Interestingly enough, the joint overlay architecture clearly benefits from system dynamics as the aforementioned issues arise only when the system is stabilized. If the system never stabilizes, which is usually the case in the dynamic environments, the cluster view and Pastry leaf set will never become static.

3.3 Design of the Joint Overlay

Each peer in the joint overlay maintains three states: a leaf set and a routing table as defined in Pastry (Section 2.1) and a cluster view as defined in the C-Gossip protocol (Section 2.2). The cluster view is composed of $c = 2^b$ semantically closest nodes, where 2^b is the base of node-IDs used in Pastry. An interest-based proximity measure is used for constructing cluster view.

In the joint overlay, the RPS-view of C-gossip is simply replaced by the leaf set and the routing table is updated opportunistically using the nodes from the cluster view. For the sake of simplicity, we assume that each node executes both maintenance protocols periodically.

Leaf set maintenance: As in the original Pastry, the leaf set is aggressively maintained. At each round, a node sends a message to all nodes in its leaf set. If a node a does not reply before the expiration of a timeout T, the node a is suspected to be failed and the leaf set is updated accordingly (the failed node is deleted and a new one is added from the closest node or the leaf set). Leaf sets are also updated upon the arrival of a new node.

Cluster view maintenance: The cluster view is updated using a gossip protocol. At each round a node chooses one node from the combination of cluster view and leaf set and

exchanges its cluster view with that node. When a node receives a view, it merges the local and received views and selects the c "closest" nodes according to the defined proximity metric. It also updates the routing table with the new received information.

4 Simulations and Results

We evaluated the joint overlay using simulations and compared its performance with the original independent overlays.

4.1 Experimental Setup

We use the *PeerSim* simulator [5] for our simulations. PeerSim is a Java based discrete-event simulator developed at University of Bologna, Italy. We used PeerSim's event-driven simulator to simulate the systems. The gossip based routing table maintenance in Pastry and RPS gossip in C-gossip is turned-off on-demand to simulate our joint overlay system. We consider a C-gossip implemented for a file sharing application, where the interest-based proximity between two nodes is determined by the overlap in their file caches. In the simulation, unless otherwise mentioned, a set of randomly selected integers are used as file identifiers. The size of the file cache is kept constant in all the nodes.

Due to the number of gossip messages generated, we had to restrict our simulations to a network size of maximum 50,000 nodes. Each simulation starts by bootstrapping the system with a single node and allowing other nodes to join in random batches. After all the nodes have joined, the simulation is run continuously for a certain number of gossip rounds in order to observe the steady state behavior of the system. To be precise, if the system takes x gossip rounds for all the nodes to complete their joins, the simulations run for $2x$ more gossip rounds after all nodes have joined. After the system has entered the steady state, various churning behaviors are applied (Section 4.3) to observe the performance of the system under churn. No search queries were generated in the system, as the chosen performance metrics (Section 4.2) do not depend on the success rate of search queries.

Each simulation compares the performances of the Pastry and C-Gossip component of the joint overlay with their stand-alone modes and with their hypothetical ideal modes. In the ideal modes, the peers are assumed to know all other peers in the system so that they can fill their node states (routing table and cluster view) as accurately as possible. The results from the ideal cases are used to provide the absolute maximum performance achievable. These ideal cases are not considered when comparing the system overheads.

4.2 Performance Metrics

The fundamental outcome of merging the overlays is the trade-off between individual overlay performances and overall maintenance overhead.

The performance of Pastry is determined by the logarithmic route complexity and this in turn is determined by the completeness of the routing table. Therefore, we consider the *number of empty cells* in the Pastry routing table as a performance measure of the Pastry protocol.

The metric *interest score* is used to measure the interest closeness of two nodes. The interest score between two nodes i and j is defined as follows: if the file caches of nodes i and j are $\mathbf{F}_i$ and $\mathbf{F}_j$ respectively, the interest score, s_{ij}, between the nodes is defined as $s_{ij} = \frac{|\mathbf{F}_i \cup \mathbf{F}_j|}{|\mathbf{F}_i|}$. For a node i with cluster view $\mathbf{V}_i$, its *average interest score*, S_i, to its view is defined as

$$S_i = \frac{\sum_{j \in \mathbf{V}'_i} s_{ij}}{|\mathbf{V}'_i|}$$

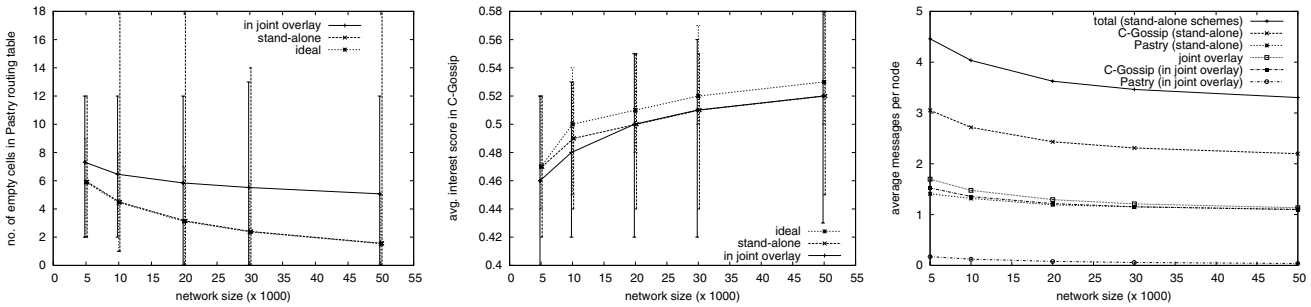
where, $\mathbf{V}'_i$ is the subset of $\mathbf{V}_i$, comprising only the nodes that are currently alive. We use this average interest score as the measure of estimating the performance of the C-gossip.

The message overhead is measured in terms of number of messages generated per node per gossip round.

4.3 Experiments and Results

Static Network First we simulated the system in a static environment where the nodes join but never leave. Figure 2(a) shows the variation of number of empty cells in the Pastry routing table according to the system size. For the sake of readibility, the y scale is centered around the results. Therefore error-bars seems important while they keep simulation reasonnably close to the average. Note that the ideal line is not a $y = 0$ line, indicating there are no candidates in the system to fill some cells of the routing tables of some nodes. This issue is reduced as the system size grows and, therefore, the average number of empty cells in the routing tables reduces with an increasing system size. A similar phenomenon can be observed on the average interest score with increasing system size.

As expected, our joint overlay shows a performance degradation, but, only by a few number of empty cells. This degradation is mainly due to the convergence of C-Gossip to a static cluster view, which in turn stops providing new candidates for the routing table entries. However, the maximum deviation of the measure is better in our system, because the Pastry routing table maintenance in our system is independent of how good the the table already is. However, the gossip protocol in Pastry depends on the entries in the routing table, which might have a negative feedback



(a) empty cells (b) interest score (c) message overhead

Figure 2. Variations of performance metrics with varying system size in a static system.

effect on filling the table. The fact that in the joint overlay, potential candidates come from another source, namely the cluster view, makes the routing tables of nodes independent and therefore insensitive to the quality of the routing tables of other nodes. Figure 2(b) shows the variation of average interest scores of the nodes with varying system size. The performance of our system is almost the same as the C-gossip with RPS gossip. The improvement of message overhead is shown in Figure 2(c). As expected, the message overhead is reduced by nearly 66%. The reduction is due to the removal of two gossip components.

Failure on a Point Time We then simulated the systems where a given fraction of nodes fails at a point in time (and remains failed) while the rest of the nodes attempts to repair the overlays. Figures 3 shows the trade-off between the performance and overhead with varying system sizes. At the middle of the simulation, 20% of the nodes crash without notice. The values shown in the figures are the ones obtained at the end of the simulation. The results are very similar to the one observed with static systems.

Figures 4 shows the results for a system of 30,000 nodes with varying failure rates. When the failure rate increases, the final system size decreases and, therefore, the results show similar performance values of smaller systems. We observe that the message overhead decreases linearly, as the "per node" value is computed by dividing the absolute value by the total system size despite the fact that the actual alive nodes are less than that. The negative feed back effect of the route gossip mentioned above is prominent (Figure 4(a)) with mass failures (75% failure) which may leave some entire rows of routing table empty.

5 Related Works

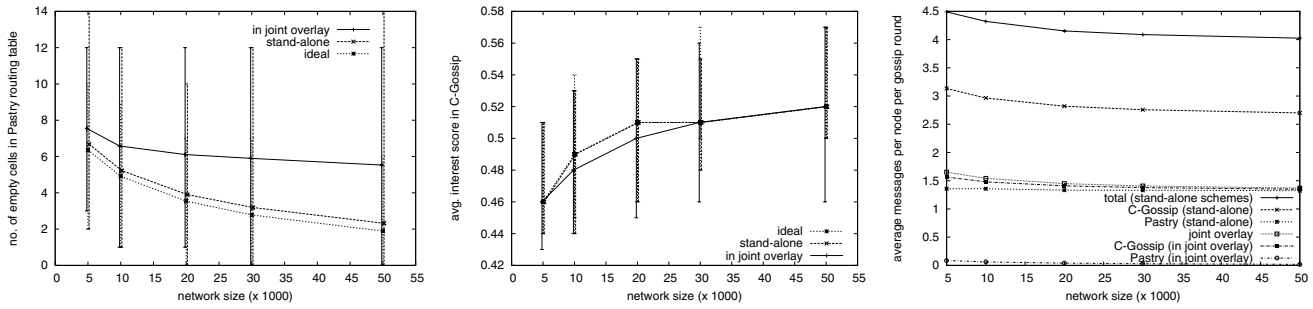
The work described in [21] proposes the use of *newscast* protocol to manage the entries in the routing tables of a structured P2P overlay. The newscast protocol is a peer sampling service [14], providing a RPS view for each node. The authors observe that the exponentially decreasing probability of filling up the routing table rows and, therefore, run one newscast instance for each row of the routing table. Each of these instances constrains its gossiping to the narrower node set to increase the probability of finding suitable candidates.

While an unstructured protocol is used above to assist the maintenance of a structured overlay, the approach presented in [16] improves the unstructured Gnutella network by adding some structural components. Even though the early Gnutella networks are completely decentralized, the latest modification includes the notion of *ultrapeers* to improve scalability. The proposed approach relies on the fact that it is more efficient to search rare items through a structured overlay and popular items through an unstructured one. The new hybrid P2P organizes the ultrapeers in a structured overlay. It disseminates the queries in the ordinary Gnutella overlay, but when it failed to receive replies within a certain time, the queries are disseminated in the structured overlay. Only the rare items are published in the structured overlay to avoid unnecessary costs.

C^2 [8] is a P2P overlay built on top of CAN [17] with a routing table notion borrowed from Chord [19]. Therefore, it provides a logarithmic routing performance as Chord, while showing enough fault tolerance capability due to its multi path routing capability achieved through the use of CAN.

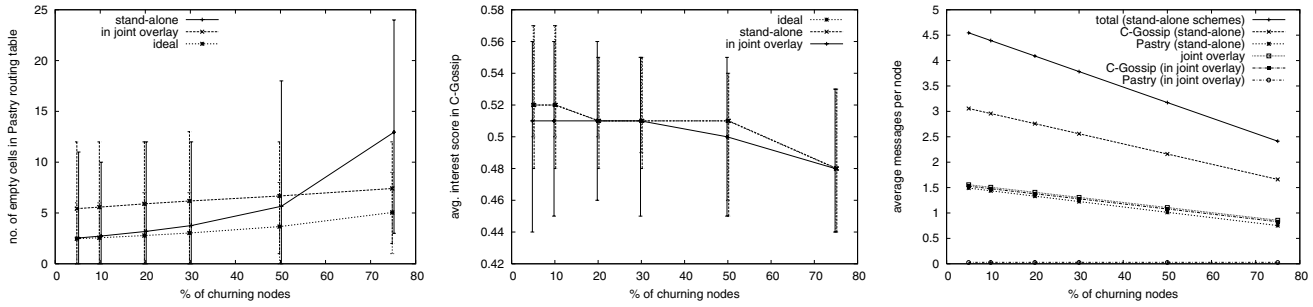
In the above mentioned works, one protocol is used to improve the other at the cost of additional overhead. Instead, we fully integrate the maintenance of both structured and unstructured overlays, resulting in a significant improvement on the overhead at the price of a slightly reduced performance.

Along the same lines as our work, at a different level, a middleware approach has been proposed to leverage the presence of multiple overlays on a same node. ODIN-S [11] presents an architecture for trading off resources between overlapping overlays.



(a) empty cells (b) interest score (c) message overhead

Figure 3. Variations of performance metrics with varying system size with failure at a point in time.



(a) empty cells (b) interest score (c) message overhead

Figure 4. Variations of performance metrics with failure rate with failure at a point in time.

6 Generalization

In this section we consider the generalization of our approach to other combinations of coexistence of P2P overlays. We believe the approach to combine structured and unstructured overlays to be generalizable with the majority of structured overlays as long as the routing information is maintained through two different sets: one primary and another secondary as identified in Section 3.1. This organization is true for the majority of structured overlays such as Chord, Tapestry, SkipNet and small world networks. CAN [17] however maintains only the equivalent of the primary component leaving no opportunity to build additional state for free. In [6], CAN is complemented with additional long links in order to improve the routing performance. We could consider this extended version of CAN for joint overlays.

Combining two structured overlays A joint overlay combining two structured overlays can be achieved in two ways based on whether the emphasize is on the routing performance or the amount of state maintained. Either the two entire systems are maintained, thus doubling the routing state on each node leaving the opportunity to greatly improving performance in both overlays, or the routing information of the two overlays may be maintained, keeping the combined size close to the size of the routing state of a single overlay at the price of a potentially reduced performance.

We did some experimental study on the first solution combining CAN and Chord. This significantly improves CAN performance (Figure 5) by providing it additional long links. The improvement is not much in Chord, because the routing complexity of Chord is already $\log(N)$. Additional links provide only marginal improvement. This shows that combining two structured overlays is not of much interest, the performance gain remaining marginal, at least for one of them, while the functionality provided by the two overlays remains similar and the maintenance overhead is doubled.

Combining two unstructured overlays Combining two unstructured overlays is very similar to the approach presented in this paper. The cluster views built according to a criterion $c1$ in one unstructured overlay may provide a random set of nodes for the other unstructured overlay clustered according to a different criterion $c2$. However, for this combination to work, the resulting clusters have to be fully independent. If there is any correlation between the two criteria, the connectivity might not be guaranteed. Consider a file-sharing system using two overlays respectively using interest proximity and geographic proximity. It is likely that French users having interests in French songs have a high probability to be geographically close as well.

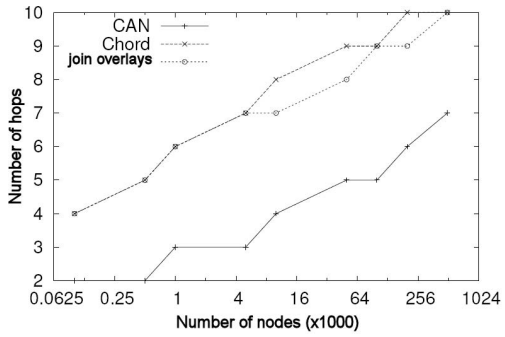


Figure 5. Variation of average search hops for 1,000,000 queries (log scale)

7 Conclusion

In this paper, we have presented the architecture of a joint overlay in which the coexistence of a structured and an unstructured overlay is leveraged. Maintaining part of each overlay provides the material to build and maintain the other part of each, thus reducing the overall system overhead without sacrificing the performances of each individual system. Our stand on this efficient trade-off between the performance and the overhead has been proven feasible by extensive simulation studies.

The system has to be further studied using real implementations and real workloads. For example, the distribution of sizes of the file caches may affect significantly the proximity measure [7]. We also plan to evaluate the impact of the joint overlay on the Pastry proximity neighbour selection. Even though we believe that proximity neighbour selection can easily be integrated in our system, the impact of the sampling size provided by the clustering protocol might affect the network stretch observed in Pastry routing. This is part of future work.

References

[1] ares. http://aresgalaxy.sourceforge.net/.

[2] Bittorrent. http://www.bittorrent.com/index.html.

[3] emule. http://www.emule-project.net.

[4] KaZaA home page. http://www.kazaa.com/.

[5] PeerSim: A peer-to-peer simulator. http://peersim.sourceforge.net/.

[6] J. Apnes and G. Shah. C2: A new overlay network based on can and chord. In *14th annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, 2003.

[7] Y. Busnel and A.-M. Kermarrec. Proxsem: Interest-based proximity measure to improve search efficiency in p2p systems. In *4th European Conference on Universal Multiservice Networks (ECUMN'2007)*. IEEE, Feb. 2007.

[8] W. Cai, S. Zhou, W. Qian, L. Xu, K.-L. Tan, and A. Zhou. C^2: a new overlay network based on can and chord. In *2nd International Workshop Grid and Cooperative Computing*, pages 42–50, Dec. 2003.

[9] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft, 2003.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8), Oct. 2002.

[11] B. F. Cooper. Trading off resources between overlapping overlays. Middleware 2006, 2006.

[12] S. B. Handurukande, A.-M. Kermarrec, F. L. Fessant, L. Massoulie, and S. Patarin. Peer sharing behaviour in the edonkey network and implications for the design of serverless file sharing systems. In *EuroSys 2006*, pages 359–372, Apr. 2006.

[13] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 79–98, Oct. 2004.

[14] M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, Oct. 2002.

[15] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster computing on the fly: P2P scheduling of idle cycles in the internet. In *The 3rd International Workshop on Peer-to-Peer Systmes (IPTPS 2004)*, Feb. 2004.

[16] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein. The case for a hybrid P2P search infrastructure. In *The 3rd International Workshop on Peer-to-Peer Systems*, pages 141–150, Feb. 2004.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *ACM SIGCOMM*, Aug. 2001.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, Aug. 2001.

[20] S. Voulgaris, E. Riviere, A.-M. Kermarrec, and M. V. Steen. Sub-2-Sub: Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks. In *The 5th International Workshop on Peer-to-Peer Systems*, Feb. 2006.

[21] S. Voulgaris and M. van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In *14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2003)*, pages 41–54, Oct. 2003.

[22] S. Voulgaris and M. van Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par 2005*, pages 1143–1152, Aug. 2005.