# Structuring Topologically-Aware Overlay Networks using Domain Names

Demetrios Zeinalipour-Yazti [a,*], Vana Kalogeraki [b]

[a] *Department of Computer Science*
*University of Cyprus*
*CY-1678, Nicosia, Cyprus*

[b] *Department of Computer Science and Engineering*
*University of California - Riverside*
*Riverside, CA 92521, USA*

**Abstract**

Overlay networks are application layer systems which facilitate users in performing distributed functions such as searches over the contents of other users. An important problem in such networks is that the connections among peers are arbitrary, leading in that way to a topology structure which does not match the underlying physical topology. This *topology mismatch* leads to large user experienced delays, degraded performance and excessive resource consumption in Wide Area Networks. In this work we propose and evaluate the *Distributed Domain Name Order (DDNO)* technique which makes unstructured overlay networks topologically-aware. In *DDNO*, a node devotes half of its connections to nodes that share the same domain-name and the remaining half connections to random nodes. The former connections achieve good performance, because the bulk of the overlay traffic is kept within the same domain, while the latter connections ensure that the topology structure remains connected. Discovery of nodes in the same domain is achieved through on-demand *lookup* messages which are guided by local *ZoneCaches*. Our technique is entirely decentralized making it appropriate for use in Wide Area Networks. Our simulation results, which are based on a real dataset of Internet latencies, indicate that DDNO outperforms other proposed techniques and that it optimizes many desirable properties such as end-to-end delays, connectivity and diameter.

*Key words:* Peer-to-Peer, Topologically Awareness, Distributed Systems.

* *Contact author: dzeina@cs.ucy.ac.cy tel: +357-22-892746, fax: +357-22-892701*
*Email addresses:* dzeina@cs.ucy.ac.cy (Demetrios Zeinalipour-Yazti), vana@cs.ucr.edu (Vana Kalogeraki).

## 1 Introduction

The advances of public networks in the last few years have increased the demand for Peer-to-Peer (P2P) application-layer protocols that can be used in the context of multicast [5], distributed object-location [24,26,27] and information retrieval [32]. Moreover, P2P file-sharing systems such as Napster [20] and Gnutella [9] have proven that large-scale distributed applications are feasible and that the P2P Computing model will play an important role in infrastructures of future Internet-scale systems.

In the P2P Computing model, participating nodes form a "virtual" overlay structure which serves as the communication medium between the participating computing units. In this model each node acts both as a client and a server, allowing users to perform distributed functions such as keyword queries. This allows these systems to harness the power of many thousands of computing units rather than only utilizing resources from a monolithic system.

P2P overlays can be divided into two categories: *Structured* and *Unstructured*. In *Structured* P2P overlays [24,26,27], network hosts and objects are structured in such a way that object location can be guaranteed within some hop count boundaries. In *Unstructured* P2P overlays on the other hand, hosts have neither global knowledge nor structure. Early unstructured systems, such as Gnutella [9], rely on flooding the network with queries in order to locate the objects. Recently more efficient query routing techniques based on routing indices [6], heuristics [30] and caching [32] were proposed.

Unstructured P2P networks offer a number of important advantages: (i) An unstructured network imposes very small demands on individual nodes, and more specifically it allows nodes to join or leave the network without significantly affecting the system performance. (ii) Unstructured networks are appropriate for content-based retrieval (e.g. keyword searches) as opposed to object identifier location of structured overlays. (iii) Finally unstructured networks can easily accommodate nodes of varying power. Consequently they scale to very large sizes and they offer more robust performance in the presence of node failures and connection unreliability.

In current unstructured systems however, the connections between peers are not based on the underlying network latencies, leading in that way to an inefficient overlay structure. This phenomenon leads to excessive resource consumption in Wide Area Networks as well as degraded user experience because of the increased network delays between the peers in the overlay network. On the other hand, the large-scale and ad-hoc nature of such systems makes it infeasible to pre-compute in a centralize setting some network-efficient overlay structure. Therefore an important problem is how to structure in a completely

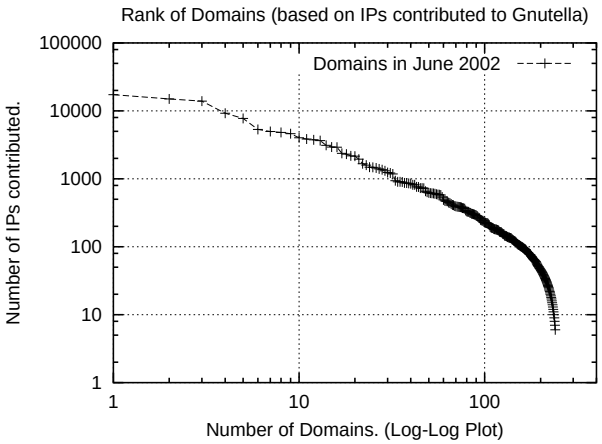


Fig. 1. Our analysis of the network traffic in [31] reveals that Large-Scale Overlay systems, such as Gnutella, consist of many thousands of nodes that belong to very few ISPs. More specifically we found that 45% of the nodes in a set of 244,000 IPs belong to only 10 large ISPs and that 58% belong to only 20 ISPs.

decentralized way an overlay network with good topological properties (i.e. low end-to-end delays, diameter and connectivity). Our motivation is to improve application performance, reduce unnecessary traffic and scale well with the size of the network.

In this work we propose and evaluate *DDNO (Distributed Domain Name Order)*, which is a distributed technique to make unstructured overlay networks topologically-aware. In *DDNO*, a node tries to connect to $degree/2$ nodes that belong to the same domain (*sibling* connections) and to another $degree/2$ of random nodes (*random* connections). The resulted DDNO topology achieves high performance through *sibling* connections while the additional *random* connections ensure that the topology structure remains connected. The choice of $degree/2$ sibling connections presents a good tradeoff between overlay performance and connectivity in networks of arbitrary degree, as we show in our experimental evaluation. Discovery of *sibling* nodes in DDNO is achieved through multicast *lookup* messages which are send out by each node and which traverse a set of *ZoneCaches* before finding other siblings. Our earlier study on the network traffic of the Gnutella [9] file-sharing network in [31], reveals that most of the participating nodes do belong to only a few ISPs (see figure 1). Therefore most nodes have a good probability of finding other *sibling* nodes which makes our scheme beneficial for the largest portion of the network. Note that these measurements are consistent with similar studies performed in 2002 by Ripeanu et. al [25], in which they found that more than 40% of these nodes are located within the top ten Autonomous Systems. Additionally the authors found that only 2-5% of Gnutella connections link nodes located within the same Autonomous System, which clearly indicates that application layer overlay networks can unnecessarily impose a huge inter-AS traffic overhead.

The DDNO overlay can become the middleware component for a variety of network-based applications. In the context of *distributed file sharing* for instance, a user in Germany has a higher probability of finding German music if his search first spans in the ".de" domains. If the overlay network is not topologically-aware, then the user's query will end up traversing domains across many different countries and continents, increasing therefore the delay of receiving back all answers and decreasing the probability of finding the desired results. Moreover, once the file is located the actual download time might also be very large as the file might physically reside far away from the user. Furthermore, our scheme can increase the performance of *P2P Information Retrieval* [32] systems. In [32] we built and evaluated a large-scale decentralized newspaper network of 1000 nodes using 75 workstations. In this context, our topologically-aware scheme will enable users to span their queries to newspaper proxies that are closer to their locations enabling them therefore to locate local news.

**Our Contribution**

In this paper we consider a fully distributed technique for addressing the problem of efficient overlay construction in unstructured networks. More specifically:

- We propose and evaluate *DDNO (Distributed Domain Name Order)*, which is an efficient, scalable yet simple technique for constructing topologically-aware overlay topologies. DDNO is entirely distributed, requires only local knowledge and therefore scales well with the size of the network.

- We provide an extensive experimental study to evaluate the performance of our technique. In addition, we compare our technique with other heuristic-based techniques. Our results indicate that *DDNO* improves many desirable properties such as low end-to-end delays, connectivity and low diameter.

The remainder of the paper is organized as follows: In section 2 we present the DDNO Algorithm, which is our proposed technique to construct topologically aware overlay networks. In section 3, we describe three alternative methods for overlay construction in centralized and distributed environments. Section 4 describes our experimental methodology, datasets and evaluation parameters. In section 5 we present our experimental results. Finally, in section 6 we discuss related work and conclude the paper in section 7.

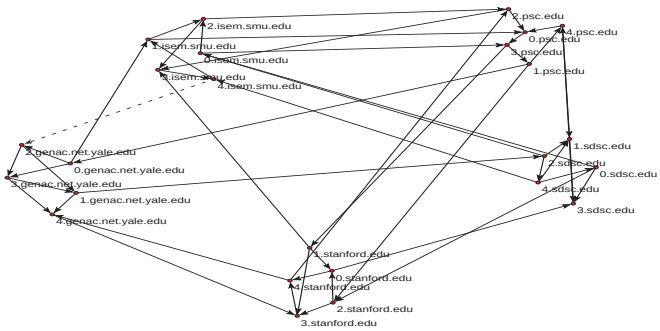


Fig. 2. A snapshot of a DDNO Topology with 25 nodes (degree=4) from 5 domains. Each node tries to connect to $\frac{d}{2}$ nodes in the same domain and another $\frac{d}{2}$ nodes in other random domains.

## 2 DDNO - Distributed Domain Name Order Protocol

In this section we present the *Distributed DNO (DDNO)* algorithm which clusters nodes belonging to the same domain together without the need of a centralized component that usually assists in the overlay construction process. In particular, we explain how nodes join the DDNO topology and how domain-name lookups are performed, with the assistance of the Split-Hash and dnMatch functions. Then, we describe the topology maintenance process and how query routing works. An example of a DDNO Topology can be viewed in figure 2. Our objective is to build an infrastructure-less protocol which in effect might be able to support large ad-hoc communities.

### *2.1 Joining a DDNO Topology*

Let $n$ denote a node which wants to join an overlay network $N$. Since $n$ doesn't know which other nodes are currently active in $N$, it has to either probe nodes to which it was connected in some past session, or to consult some distributed discovery service $D$ (i.e. some *hostcache*) which will provide $n$ with an initial list of active nodes[1]. We assume that an out-of-band discovery service will provide $n$ with a random list of active hosts $L=\{n_1, n_2, ..., n_k\}$, for some constant $k \geq \frac{degree}{2}$. It is important to note that the individual hostcaches do not have global knowledge and therefore these cannot be used for disseminating some pre-computed overlay structure or the distances between all node pairs.

After $n$ obtains the list $L$, it first attempts to establish a connection to $d/2$ random nodes, where $d$ is the degree of $n$. The pseudocode of this procedure

[1] Both techniques are deployed in many Peer-to-Peer systems, such as Gnutella [9] and Kazaa [15] and work reasonably well. Hostcaches are either located on Web pages or dedicated servers.

**Algorithm 1** Join_Network

```
1: procedure JOIN_NETWORK(n, N)
2:     random ← 0
3:     while true do
4:         while (random < d/2) do
5:             if (empty(L)) then
6:                 L ← getRandomList(d/2)
7:             end if
8:             random ← connect(L)
9:         end while
10:        next ← getRandNode()
11:        L ← lookupDN(dn(n), ttl, next)
12:        wait(interval)
13:        connect(L)
14:    end while
15: end procedure
```

can be viewed in lines 4 to 9 of Algorithm 1. In the experimental results of Section 5.3, we validate the choice of $d/2$ random nodes and show that it presents a good tradeoff between overlay performance and connectivity in networks of arbitrary degree.

Note that in the procedure of Algorithm 1, it is quite possible that some or all of the nodes $n_i$ in $L$ are not able to accept any new incoming connections. This might either happen because $n_i$ reached its maximum degree or because $n_i$ went offline. It this case $n$ will need to obtain an additional list $L$ from $D$ and repeat steps 4-9 until $d/2$ random nodes are found. In the next step we attempt to find $d/2$ sibling connections by sending a `lookupDN` message to one of the existing (random) neighbors. The message will attempt to return a number of sibling nodes in $N$. We will discuss the complete operation of the `lookupDN` message in subsection 2.3. Since a `lookupDN` message might get terminated without returning any results, a node might pipeline several such messages.

Before describing in further detail the *lookupDN* message, we will describe two useful functions: i) Split-Hash, which allows us to efficiently encode domain names and ii) dnMatch which determines whether two domain names $dn_1$ and $dn_2$ belong to the same domain or not.

### *2.2 The Split-Hash and dnMatch Functions*

Each node participating in a DDNO topology has some **Domain Name (dn)**, which is a string that conforms to the syntax rules of RFC 1035 [19]. Such a string, which is case insensitive, can be expressed with the regular expression

$dn = label(.subdomain)^+$, where label and subdomain are some strings with certain restrictions, such as length and allowed characters. In order to determine whether two domain names $dn_1$ and $dn_2$ belong to the same domain we first introduce the *split-hash* function, which is a hashing function that splits a domain name $dn$ into $k$ hashes, where $k$ is the number of subdomain strings in $dn$ ($k = |subdomain(dn)|$). A formal definition of this function is given in the *split-hash* procedure.

```
1: procedure SPLIT-HASH(dn)
2:     int size = |subdomain_dn|
3:     for j = 1 to size do
4:         result[j] = hash(m, subdomain_dn[j])
5:     end for
6:     return result
7: end procedure
```

In the procedure, $hash(m, subdomain_{dn}[j])$ hashes the $subdomain_{dn}[j]$ using $m$ bits. We chose to use hashcodes instead of raw domain-names because it allows us to keep the lookup message size small [2]. Furthermore, for performance reasons the hashcode does not need to be a non-colliding key [3] as this would again make $\ell$ prohibitively large. For example if we use a total of 160 bits for all the $k$ generated hashes, then there would be an additional 100 bytes augmented to the lookup message after 5 hops. Instead, using a 20-bit hash function and assuming that keys are uniformly generated, we will be able to uniquely identify more that 1 million nodes and travel a distance of 40 hops with the same amount of bytes.

Now that we have introduced *split-hash*, we use the **dnMatch**$(dn_1, dn_2)$ comparison function, which compares the individual hashes of *subdomains* $dn_1$ and $dn_2$. In the basic case, **dnMatch** returns *true* if $dn_1 \neq dn_2$ and the subdomain of $dn_1$ and $dn_2$ matches. For example if $dn_1$="a.aol.com" and $dn_2$="b.aol.com" then dnMatch$(dn_1, dn_2) = true$. For $dn_1$="a.yahoo.de" and $dn_2$="a.yahoo.com" then dnMatch$(dn_1, dn_2) = false$. Of course our scheme can take advantage of the hierarchical structure of DNS and return the amount of similarity between two domain names (instead of using an exact match answer). For example if $dn_1$="a.rochester.rr.com" and $dn_2$="b.ny.rr.com" then dnMatch can return $\frac{2}{3} = 0.66$, rather than simply true or false.

The only limitation with *dnMatch* is that it can't distinguish two nodes that share the same *dn*, such as nodes in private networks using NAT (Network Address Translation). Although these nodes won't be able to connect to each other as siblings, they present only a small fraction of the nodes in networks such as Gnutella, in which they are less than 5% [31].

[2] RFC 1035 [19] defines that subdomain name must be 255 characters or less.
[3] Hash functions such as SHA-1 are 160-bit and collision of two keys is difficult.

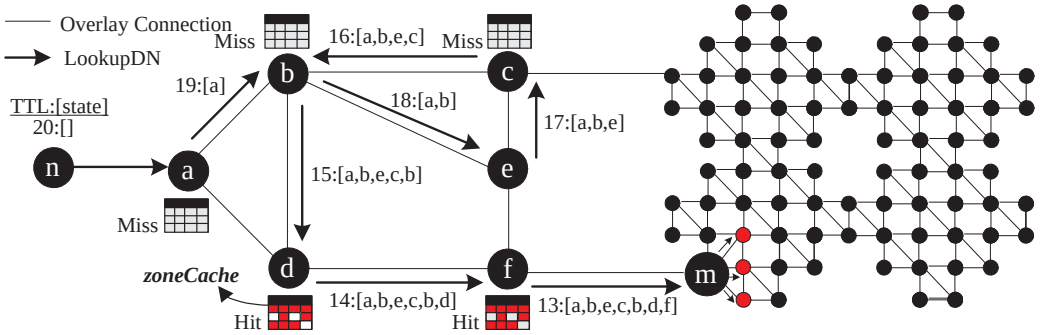


Fig. 3. **Domain-Name Lookup in a DDNO topology.** Each *lookupDN* message retains path information in order to populate the *ZoneCaches* of other nodes. The list appends shown on the lookupDN message in the picture illustrate the accumulated path in $\ell$.

### *2.3 Domain-Name Lookup in a DDNO Topology*

We now focus our attention on the *lookupDN* procedure which is used by some node $n$, in order to discover other *sibling* nodes in $N$. We model the lookupDN message (denoted as $\ell$) as a *multicast walker*. The goal of the multicast walker $\ell$ is to reach some node $d$ that can guide it to the destination (i.e. a sibling of $n$). Note that before reaching $d$, $\ell$ may need to traverse a number of randomly selected neighbors. This can be viewed in figure 3, in which $\ell$ takes the random itinerary $[a, b, e, c, b, d]$. At $d$ however, $\ell$ is allowed to make an informed decision on which neighbor to follow next (in this example node $f$). This is achieved by using a special structure called *ZoneCache* that contains information on which nodes are reachable in a $r$-hop radius (it will be discussed in the next subsection). At the end of this procedure, $\ell$ is expected to reach some node $m$, which is a sibling of $n$. $m$ then issues a *broadcast* message to all of its own *siblings*. Each of the receiving nodes, including $m$, will respond with a *LookupOK* message (denoted as $\ell'$) if they are willing to accept new connections. Therefore node $n$ will end up receiving several answers out of which it will attempt to establish a connection to $degree/2$ nodes.

If the nodes along $\ell$'s path don't keep any intermediate state (e.g. the identifier of $\ell$ [4]), then $\ell$ might get locked in a cycle (e.g. loop $b \rightarrow e \rightarrow c$ in figure 3). Instead of retaining such state at each node, we choose to incorporate state information in $\ell$ as this also serves as an implicit mechanism to populate the ZoneCaches along $\ell$'s path. The state information included in $\ell$, includes the **split-hash** $h$ (described in section 2.2), on the domain-name of each node that $\ell$ traversed (i.e. $state_\ell = \{h(v_n), ..., h(v_m)\}$).

[4] $\ell$ messages are uniquely identified by a random identifier that is locally generated by the issuer of $\ell$.

**Algorithm 2** lookupDN

```
1: procedure LOOKUPDN(n, ttl, m)
2:     cacheRoute(n,ZoneCache(m))
3:     if ( dnMatch(n, m) ) then
4:         if ( (degree(m) < d) and not(connected(n, m)) ) then
5:             send(n,"LOOKUPOK m.IP,m.PORT")
6:         end if
7:         broadcast(siblings(m))
8:     else if (ttl > 0) then
9:         if ( hit( ZoneCache(m), hash(n) ) ) then
10:            next ← getNextNode(ZoneCache(m))
11:        else
12:            next ← getRandNode()
13:        end if
14:        lookupDN(n, ttl − 1, next)
15:    end if
16: end procedure
```

The complete pseudocode of the *lookupDN* procedure can be viewed in Algorithm 2. A node $n$ sends a lookup message to node $m$ using some *ttl* parameter, which determines the maximum number of hops that the given lookup should be forwarded. The *ttl* parameter, which is used in many networked applications, starts out from a predefined value and is decrement each time a lookup message is forwarded until it becomes zero.

### *2.4 The ZoneCache Structure*

*ZoneCache* is a caching structure which is deployed locally at each node and its functionality is to guide `lookupDN` messages to their sibling nodes. A snapshot of such a structure is displayed in table 1. The first column includes the hash of some domain-name and this information is extracted from passing lookupDN messages. The second column indicates, the peer connection that will lead a future $\ell_2$ to the corresponding destination, and the third column indicates the respective cost in hops. Finally ZoneCache also uses a timestamp parameter (fourth column) in order to limit the number of entries to a total size of $C$.[5] Once the repository of some node becomes full the node uses the Least Recently Used (LRU) policy to keep the most used entries in the cache.

The cache stores only the hashcodes of the nodes that are located within an $r$-hop radius in order to limit its size and accuracy. We show how this works with the following example: Assume that node $n$ sends a lookupDN message $\ell$ searching for some sibling and that this message reaches some node $d$ (in figure 3). Also assume that $\ell$ has already passed from five nodes and that

[5] We set ZoneCache's maximum entries parameter $C$ to 350.

Table 1
The **ZoneCache Structure**. It caches domain reachability information from lookupDN messages that traverse a given node.

| **Split-Hash** | **Neighbor** | **Hops** | **LRU TimeStamp** |
|---|---|---|---|
| 9A78DF | Socket3 | 3 | 10000000 |
| 421CDE | Socket1 | 2 | 10012000 |
| ... | ... | ... | ... |
| 2AB356 | Socket1 | 2 | 10160000 |

it has the following state: $state_\ell = \{a, b, e, c, b\}$. If the radius parameter of $m$'s ZoneCache is set to three then node $d$ will store the following quadruples (i.e. information for only three hops away): $\{(b, b, 1, ts)$, $(e, b, 2, ts)$, $(c, b, 3, ts)$, $(a, b, 2, ts)\}$, where the first field is a hash of the destination node, the second field the next neighbor that leads to the destination, the third field the number of hops and the last field the timestamp parameter generated at the time of the record insertion.

Note that before storing the quadruples, we identify and eliminate cycles in the $state_\ell$ sequence (therefore $(a, n, 2, ts)$ is also considered). Furthermore if $d$'s ZoneCache already contains any of the following hashcodes $\{a, e, c, b\}$ then $\ell$ would update some tuple only in the case that the new entry provides a shorter path to the respective entry. The next question is how the cached information becomes useful to some future lookup message. Suppose that node $a$ sends a lookupDN message $\ell_2$ to $d$ (see figure 3) and that $a$ and $c$ are siblings (i.e. $dnMatch(a, c) = true$). Following the previous example, $d$ has an entry in its zonecache which indicates that $c$ can be reached through $b$ in 3 hops. Therefore $\ell_2$ will be routed towards $c$. Although neighboring ZoneCaches could actively exchange routing updates at regular intervals, like BGP, our passive caching scheme reduces significantly the amount of transmitted message and works well in dynamic environments as we will see in section 5.

### *2.5 DDNO Topology Maintenance*

When a node disconnects from the DDNO topology it does not need to send any a priori notification to the other nodes. This happens because each node continuously tries to maintain its degree to the pre-determined value $d$. If some *random* neighbor of $n$ leaves $N$ then $n$ will either attempt to re-establish the dropped connection or find another node from the discovery service outlined in subsection 2.1. On the other hand, if some *sibling* of $n$ disconnects then $n$ consults its *ZoneCache* in order to send the new lookupDN message towards a current sibling. It is expected that $n$ will discover another sibling in only two hops (as a node already maintains $(\frac{d}{2} - 1)$ siblings).

Another technique would be to proactively exchange `lookupDN` messages with sibling nodes. Although this might allow a node to instantly react in the event of failures, it might become a large overhead for the overlay topology. For example our study in [31], on a collection of 56 million overlay messages obtained from the Gnutella network, reveals that 23% of all messages are `Ping` messages and 40% of them are `Pong` messages. `Ping/Pong` messages are the main technique for proactively discovering new nodes in the Gnutella Network.

### *2.6 Query Routing in a DDNO Topology*

One of the major objectives of overlay networks is to facilitate users in performing distributed functions, such as queries over the contents of other users. In [32], we have made an extensive study on a number of different query techniques that can be applied in randomly generated topologies. In this work we propose the deployment of the DDNO topology which leads to more desirable overlay properties. Given that we have a DDNO topology some node might deploy any of the following search techniques: *Breadth-First-Search (BFS)* [9] (query all neighbors), *Random BFS* [13,32] (query a random subset of neighbors), *ISM* [13,32] (intelligently query a subset of neighbors) or $>$*RES* [30] (query the neighbors that returned the most results in the past). Our study which was performed on a real network of 1000 nodes deployed on a network of 75 PCs, reveals that by using our ISM technique we might be able to retain high degrees of recall while using only half messages and time used by the brute-force BFS technique.

DDNO allows multiple search algorithms to be deployed on top of its topology. The advantage of using DDNO is that the bulk of the incurred overlay traffic will remain within the same domain since only $d/2$ of the traffic will make its way to a different domain. Finally, the DDNO topology gives space for more sophisticated search techniques. In the context of a large-scale file-sharing application with many thousands of nodes, we might decide to forward query requests to only *sibling nodes*.

### *2.7 DDNO in a Hybrid Overlay Environment*

Although the proposed DDNO topology leads to a flat topology, the basic approach can be utilized in some hybrid P2P environment such as Kazaa[15] and Gnutella[9] v0.6. In such an environment some nodes with long-time network connectivity and high bandwidth connections, known as *SuperPeers or UltraPeers*, form a backbone infrastructure which can be utilized by other less powerful nodes (denoted as *RegularPeers*). Such a model allows the network size to grow to millions of users because it differentiates short-time connection

and modem users from other more powerful users (e.g. ADSL, cable modem users).

DDNO could be deployed in a hybrid P2P environment in the following way: A superpeer $s$ initiates a lookupDN $\ell$ message to find $d/2$ other sibling and $d/2$ random superpeers. RegularPeers will again utilize the lookupDN message to discover the superpeer nodes that belong to their domain and that might be able to serve them. Of course using such an approach in an overlay, requires a large number of participating nodes, as smaller numbers would limit the number of superpeers the $\ell$ message locates. Therefore in our experimental evaluation of section 5, we use the basic "flat" topology approach, rather than the "hybrid" topology discussed in this subsection.

## 3 Alternative Heuristics for Overlay Construction

In this section we will describe various overlay construction heuristics that are later compared to DDNO. We start out by defining the computation model of these algorithms. Specifically, each algorithm takes as an input a vertex set $V = \{1, 2, ..., n\}$ and constructs an overlay topology $G = (V, E)$, where the $E$ set represents the overlay connections between the $V$ vertices. The construction of an optimal overlay is known to be NP-complete [8] therefore the following presented algorithms are, similarly to DDNO, based on heuristics.

To simplify the discussion, we start out by describing the functionality of each algorithm using a centralized setting. In a centralized setting, a global list of all $n$ nodes along with physical distances between all pairs (the $n$x$n$ IP-latency adjacency matrix), is known to every node in the system. We then formalize how each of these heuristics can be applied in a distributed manner. Note that the centralized version of each algorithm is not only useful for describing the distributed techniques, but also provides us with a lower bound on their overlay performance. This is attributed to the fact that the centralized version of each algorithm can utilize global information in order to find the best peers among its possible choices. The lower bound allows us to know the "best-case" overlay performance for each of the described distributed techniques.

### *3.1 The Random Algorithm*

In this algorithm, each vertex $v_i$ selects its $d$ neighbors by randomly choosing $d$ other vertices. Since overlay connections are bi-directional, a node avoids connecting $v_i$ to $v_j$ if $v_j$ is already connected to $v_i$. This is the algorithm deployed in most current P2P networks such as [9,15] and its main advantages

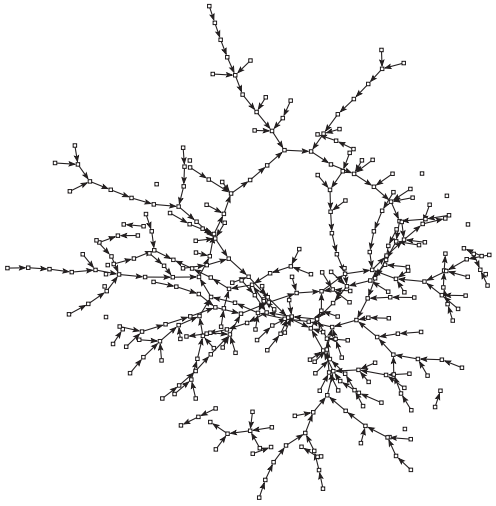

Fig. 4. Visualization of a Random graph with n=332 nodes (average degree=2, diameter=32) using the Kamada-Kawai visualization model in Pajek [1]. Random topologies have the advantage that they are easy to construct and lead to connected topologies (if $degree > log_2 n$ [2]). The latencies at the overlay-layer however, usually don't match the underlying physical latencies.

are: i) it is simple as it does not actually require the $n$x$n$ IP-latency matrix (therefore it can be completely distributed), and ii) it leads to connected topologies if the degree $d > log_2 n$ [2] (see figure 4). We will refer to the centralized and distributed version of this algorithm as **RAN** and **DRAN** respectively.

### *3.2 The Short-Long and BinSL Algorithm*

The centralized **Short-Long (SL)** algorithm, which was proposed in [23], alleviates the network unawareness of the RAN algorithm in the following way: Each vertex $v_i$, selects its $d$ neighbors by picking the $d/2$ nodes in the system that have the shortest latency to itself (these connections are called *short links*) and then selects another $d/2$ vertices at random (these connections are called *long links*). Assuming that the SL algorithm has at its disposal the $n$x$n$ IP-latency matrix, it can easily find the latencies between the various node pairs. The intuition behind this algorithm is that the $d/2$ connections to "close-by" nodes will result in well-connected clusters of nearby nodes, while the random links serve to keep the different clusters interconnected and the overall graph connected.

Constructing and maintaining the $n$x$n$ IP-latency matrix in a large scale environment is very difficult. This is particularly true, in the presence of a high churn rate [4]. Therefore Ratnasamy et. al also propose the distributed **BinSL**

**Algorithm**, which utilizes the notion of *distributed binning* in order to approximate, in a completely distributed fashion, the latencies between nodes.

More specifically each node calculates the round-trip-time (RTT) from itself and $k$ well-known *landmarks* $\{L_1, L_2, ..., L_k\}$ on the Internet. The set of landmarks can consist of stationary entities (such as DNS servers). The numeric ordering of the latencies represents the "bin" the node belongs to. Latencies are then further classified into *level* ranges. For instance if the latencies are divided into 3 levels then; *level 0* accounts for latencies in the range [0,100), *level 1* for the range [100,200) and *level 2* for all other latencies. The level vector is then augmented to the landmark ordering of a node yielding a string of the type "$l_2 l_1 l_3 : 011$". An example execution of this heuristic can be seen in figure 5 (left).

It is expected that nodes belonging to the same bin will be topologically close to each other although *false positives* are possible, that is, some nodes do belong to the same "bin" although they are not topologically close to each other. The rate of *false positives* is a function of how many landmarks are used, as fewer will degrade the performance of the binning scheme. For example if two nodes, the one located on the east coast and the other the west coast, have the same RTT to k landmarks, then they will also share the "bin" code. In our experiments, presented in Section 5, we experimented with the following scenarios: i) **BinSL-4** which uses 4 landmarks with 3 levels and ii) **BinSL-12** which uses 12 landmarks and 3 levels.

### *3.3 The Greedy Binning Algorithm*

In order to emphasize that by only selecting the shortest latency nodes might have a negative effect for the overall network structure; we also propose and study the Greedy Short (**Short-Short or SS**) algorithm. In $SS$, each vertex selects as its $d$ neighbors the ones that have the shortest latency to itself (i.e. only short links). Our experimental study in 5, reveals that this always results in disconnected topologies (this is also visualized in Figure 5 (right)).

### *3.4 Domain-Name Order Algorithm (DNO)*

In the Domain-Name Order Algorithm, which is the centralized version of the algorithm we propose in section 2, a vertex $v_i$ selects its $d$ neighbors by picking the $d/2$ vertices that have the same domain-name with $v_i$. It then selects another $d/2$ neighbor at random. The idea of this algorithm is similar to the *SL* algorithm, in that we want to create well-connected clusters of nodes that are topologically close to each other without jeopardizing network

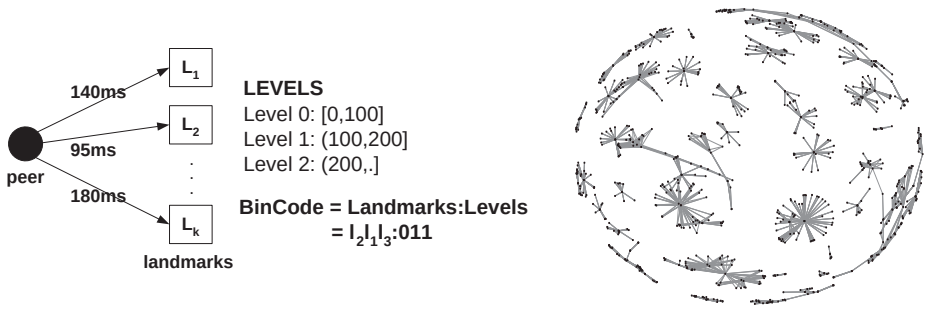


Fig. 5. a) An example execution of the BinSL Algorithm. b) Visualization of Greedy Binning (SS) graph of 1000 peers (degree=6) using the Fruchterman-Reignold visualization model in Pajek [1].

connectivity. The main advantage however, is that the *DNO* algorithm does not require the $n$x$n$ IP-latency matrix that $SL$ needs.

## 4 Experimental Methodology

Our experimental evaluation focuses on: (i) the *Overlay Performance*, in which we evaluate the generated overlays with respect to the overall end-to-end delays, the graph diameter and the number of clusters, and ii) *Lookup Performance*, in which we evaluate the performance of lookupDN messages with respect to the number of hops each message traverses and the percentage of resolved queries.

### *4.1* Evaluation Parameters

**Overlay Performance:** To assess the performance of an overlay we define the Aggregate All-Pair Shortest Path (AggAPSP) parameter, which is the sum of all distances (pairs of shortest paths) on the overlay graph $G$. Formally $AggAPSP$ is defined as following:

$$AggAPSP_G = \sum_{i=0}^{n} \sum_{j=0, i \neq j}^{n} APSP[i][j], (APSP[i][j] \neq \infty) \qquad (1)$$

where $APSP$ is an $n$x$n$ matrix that stores all the minimum distances between all pairs. Such a table is obtained by running some All-Pair Shortest Path (APSP) algorithm [6] on the set of pairs in the edge set $E$. $AggAPSP$, can be thought as the end-to-end delay between all different pairs, and that is the

[6] We use the Floyd-Warshall Algorithm [10].

reason it needs to be minimized. Although routing of messages on an overlay is performed based on the routing policies defined by each node, we use shortest path routing (similarly to [23]) which provides lower bounds for paths taken by packets.

Formula 1 however, does not take into consideration the fact that some connection between overlay nodes might not be available. This happens in the case that the overlay network $G$ is segmented into two or more partitions. Therefore we also define the $Clusters_G$ metric, which is the number of disconnected groups of nodes a given graph has. More formally:

$$Clusters_G = COUNT(Connected\_Components) \tag{2}$$

where $Connected\_Components$ is an algorithm that identifies the connected components of a graph [7]. It is important to mention that disconnected network segments are undesirable in overlay networks as this limits the reachability of nodes in the network.

Finally, we also take into account the *Diameter* of an overlay $G$, which is the length of the longest of shortest path distances between pairs $v_i$ and $v_j$ $(\forall i, j \in V)$. More formally $Diameter_G$ is defined in the following way:

$$Diameter_G = MAX(SP(v_i, v_j)), (\forall i, j \in V \text{ and } i \neq \text{j}) \tag{3}$$

where SP is the maximum shortest path between vertices $v_i$ and $v_j$. Consider for example two overlay instances $G_1$ (ring topology) and $G_2$ (star topology) with the same number of vertices that have only different diameters $\delta_1$ and $\delta_2$ $(\delta_1 \gg \delta_2)$. If an overlay message uses a parameter `TTL`, which limits the number of hops a message travels, then the nodes reached by the message are much less for $G_1$ than $G_2$. Therefore large diameters play a negative role in the resolution of some overlay message (e.g. some `QUERY` message) as those messages are required to travel more hops and possibly won't reach an adequate number of receivers.

**Lookup Performance:** Since the functionality of lookupDN messages is of major significance in the context of DDNO, we investigate the average number of hops each lookupDN message $\ell$ traverses before finding some sibling node. We also investigate the total number of temporary connections that are swapped with sibling connections once the latter are found under various scenarios of churn. Note that in DDNO a node attempts to connect to degree/2 nodes in the same domain (siblings) and degree/2 random nodes. However if degree/2 siblings are not found, then each node temporarily utilizes the random nodes until the requested amount of siblings is located.

[7] We use the Component-Finding algorithm that uses DFS [10].

### *4.2* **Description of Datasets**

Evaluating topologies based on the parameters outlined in the previous subsection requires a dataset in which the IP latencies are not synthetic. We therefore chose to base our experiments on the measurements of the *Active Measurement Project (AMP)* [11], at the *National Laboratory for Applied Network (NLAR)*. AMP deploys a number of monitors distributed along 130 sites to actively monitor the Internet topology. AMP monitors ping and traceroute each other at regular intervals and report the results back to the project servers. Most of the current 130 monitors currently reside in the U.S with a few exceptions of some other International sites. The details of the AMP methodology and infrastructure can be found in [11].

In our experiments we use a 1.8 GB snapshot of traces obtained on the 30th of January 2003. The set includes data from 117 monitors out of which we extracted the 89 monitors which could be reversed DNS (i.e. given their IP we obtained a DNS name). We then construct the $n$x$n$ IP-latency matrix (for all n=89 physical nodes), that contains the latency among all monitors. Since all 89 hosts are located at different domains, we choose to incorporate some degree of host replication per domain. Our study in [31] shows that hosts in a real overlay network, such as Gnutella, exhibit this characteristic. More specifically we choose the following host replication schemes:

(1) **Random Replication (RR)**. We randomly replicate each host $[1..k]$ times. In our experiments we set $k = 7$ which consequently generated 332 nodes. This network attempts to address scenarios in which some domains contribute more hosts than other domains.
(2) **Uniform Replication (UR)**. We replicate each host $k$ times, for some parameter $k$. In our experiments we set $k = 4$, which consequently generated 356 nodes. This network attempts to address scenarios in which all domains contribute equally to the host distribution of the network.

Additionally, for the scalability experiments presented in Section 5.6, we derive two new datasets from the Active Measurement Project. More specifically we chose the Random Replication scheme and generate the **Large Random Replication (LRR)** dataset with 5,000 (**LRR-5K**) and 10,000 (**LRR-10K**) nodes.

### *4.3 Time Model*

Since there is no fine-grained model of time in a simulation environment, we choose to divide time into units of algorithm *iterations*. During an *iteration* each node $n$ is given the opportunity to establish connections to up to $d$

neighbors. $n$ is not assured that it might be able to connect to $d$ neighbors in a single iteration. This happens because some or all of its attempts target nodes that have already reached their expected degree and therefore don't accept any new incoming connections. Therefore an algorithm may require several iterations before it stabilizes.

## 5 Experimental Evaluation

In this section we present the results of our extensive experimentation with DDNO. More specifically, we implemented centralized and distributed versions of the various algorithms presented in Section 2 and Section 3. Note that in a distributed setting some node has no topological information other than which are its own neighbors. Therefore global lists of other active nodes or IP-latencies are not available.

### *5.1 Centralized Evaluation*

For the first experimental series we run the four centralized algorithms *RAN, SL, SS and DNO* using both the RR and UR datasets. The presented results are for different degree parameters larger than five (at which most algorithms stabilize to a single cluster). In Figure 6 (top row) we can see that all algorithms, other than $SS$, have a large $AggAPSP$ value for smaller degrees but as the degree increases, AggAPSP quickly stabilizes. This happens because initially there are fewer paths between the different node pairs. For example, if we have three nodes $a, b, c$ connected in the following topology $a \overset{5}{\leftrightarrow} b \overset{10}{\leftrightarrow} c$ (where the number on the edges represents the latency between the respective nodes), then the addition of edge $a \overset{2}{\leftrightarrow} c$ will drop the $AggAPSP$ from 30 (5+10+15) to 14 (5+7+2). $SS$ on the other hand, presents always a very low $AggAPSP$ because: i) each node only chooses the nodes that have the shortest latency to itself and ii) because the network topology is always disconnected and therefore many entries are not considered (i.e. APSP[i][j]=$\infty$). The figure indicates that RAN has the highest $AggAPSP$, which means that it has the highest end-to-end delay between nodes. $SL$ and $DNO$ on the other hand are both able to perform much better because both algorithms choose half of their neighbors selectively, (i.e. the lowest IP latency and domain-name match respectively). On the same figure we can see that $SL$ performs slightly better than the DNO algorithm but this is expected as $SL$ has the advantage of choosing the $d/2$ "least latency" nodes while $DNO$ has to rely on the domain-names as a metric for network distance. $DNO$ however doesn't utilize the IP-latency table which provides the latencies between all pairs.

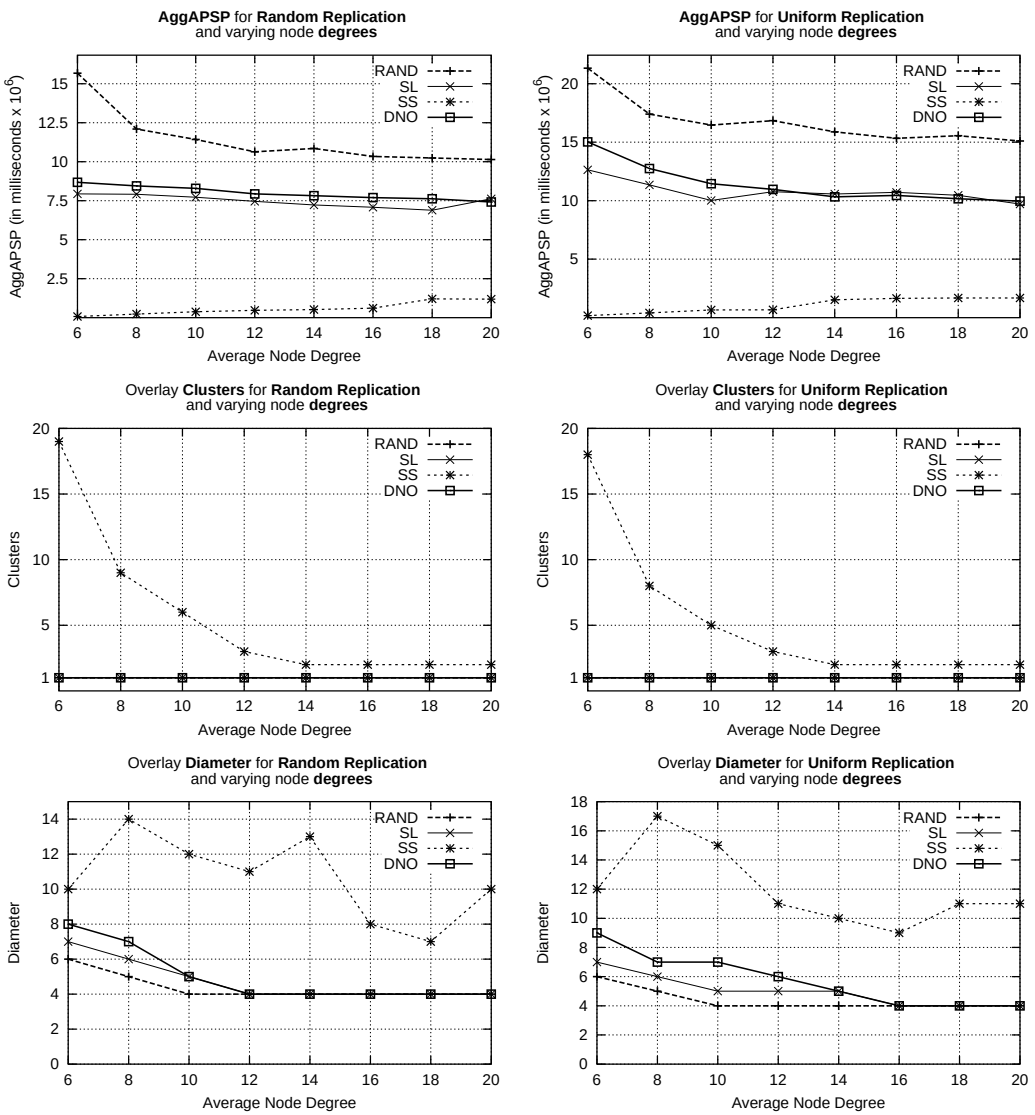


Fig. 6. Evaluating overlay topologies using data from the RR (left) and UR (right) datasets. By using $DNO$ or $SL$ we can significantly reduce the end-to-end delay between overlay nodes (top), while maintaining a connected topology (middle) with a relatively low *diameter* (bottom) for the same average degree.

The fact that only $RAN, SL$ and $DNO$ generate connected topologies can be observed in Figure 6 (middle row). More specifically all three algorithms yield connected topologies while $SS$ always results in disconnected topologies even for very large degree values (i.e. 20). This happens because each node selects as its $d$ neighbors only the nodes that have the shortest latency to itself.

Figure 6 (bottom row) shows the *diameter* of the four algorithms. As we can see only the $SS$ algorithm generates topologies with arbitrary large diameters even in the case of very large degree parameters. On the same figure we can see that $SL$ again slightly outperforms the DDNO algorithm but only for smaller degree parameters (less than 10). This is again expected as SL optimizes more

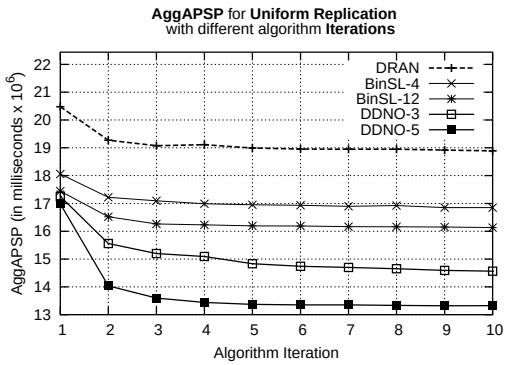


Fig. 7. **Performance Evaluation** of DRAN, BinSL-4, BINSL-12, DDNO-3 and DDNO-5 using the UR dataset

the latency parameter (shortest path) between nodes, which consequently also decreases the diameter of the network. All experiments are averages of five executions.

In this subsection we have seen that DNO might be useful for constructing connected overlays with low end-to-end delays and low diameters. The problem that arises in a real overlay setting is that we don't have global knowledge or not even a list of active nodes at all times. One solution would be to deploy some centralized lookup service, that, given some domain-name, returns IPs of other active nodes that belong to the same domain. This service might pre-compute some overlay structure, similarly to [14,16], and disseminate it back to the nodes that participate in a given overlay. Although such services are feasible, they are expensive, usually don't scale well and are vulnerable to denial of service attacks and censorship [20].

### *5.2 Performance Evaluation*

In the next experiment we evaluate the distributed algorithms against the three parameters we defined in section 4.1 (i.e. AggAPSP, Clusters and Diameter) using the *UR* dataset in which each node has a degree of 8. We choose this parameter as it always results in connected topologies. For our Distributed Domain Name Order Algorithm (DDNO), we advocated in section 2, we have experimented with various parameters for the *caching radius* of the ZoneCache and for the *ttl* parameter of lookup messages. Here we present our most representative configurations which are summarized as following: i) **DDNO-3** which uses lookupDN messages with a ttl of 20 and caching radius of 3. ii) **DDNO-5** which again uses a ttl of 20 but caches in a radius of 5.

As we can see in figure 7 DRAN has again the highest end-to-end delay as the AggAPSP stabilizes at 19M ms while all other algorithms perform much better. DDNO-3 and DDNO-5 use $\approx 13.5 - 14.5$M ms while BinSL-4 and BinSL-12

use $\approx 16 - 17$M ms. This means that DDNO-5 presents a 30% improvement upon the DRAN technique. We can also see that although we increase by three times the number of landmarks in the BinSL algorithm the accuracy of the binning scheme only increases about $0.8M$ ms.

We can also observe that DRAN and BinSL manage to stabilize within the first few iterations as their operation doesn't involve temporary connections. We already discussed that DDNO maintains more that $d/2$ random connections if it is not able to locate $d/2$ siblings. Although this increases connectivity and prevents network fragmentation, it also slightly delays the stabilization process. We also experimented without allocating temporary connections and found that such an approach is viable, as it stabilizes after the seventh iteration, but it initially results in a very high AggAPSP. In this experimental series all algorithms always generate connected topologies which therefore make the $Clusters_G$ evaluation parameter equal to one. Furthermore the $Diameter_G$ remains constant at five. Therefore the graph for both evaluation parameters is omitted.

### *5.3 Sibling Factor Evaluation*

So far, in the experiments with DDNO we have used a *sibling factor (K)* of $degree/2$. In this section we experimentally evaluate the DDNO algorithm using a varying parameter $K$ in both dense and sparse networks. The evaluation in this section demonstrates that by increasing the sibling factor in a densely connected graph (degree=8), the AggAPSP parameter linearly decreases, while retaining a connected topology. On the other hand, our evaluation in this section also demonstrates that by repeating the same experiment in a sparsely connected topology (degree=4), results in highly disconnected topologies.

Specifically, figure 8 (left) shows the AggAPSP parameter for the connected topology using the *UR* dataset. The figure indicates that by increasing the K parameter the performance of the overlay structure increases (i.e. smaller AggAPSP). In the figure we can also observe that for a sibling factor of K=degree (i.e. 8), we can achieve a 12M AggAPSP latency, which is close to the 8M AggAPSP lower bound presented in section 5.1. Note that for all experiments presented in this paper the TTL parameter of the lookupDN was 20. Had we chosen a larger parameter it would be possible to optimize the overlay even further.

In the next experiment we measure the performance of the algorithms in a sparse topology (degree=4). Note that we cannot directly measure the AggAPSP parameter when the graph gets disconnected. This happens because

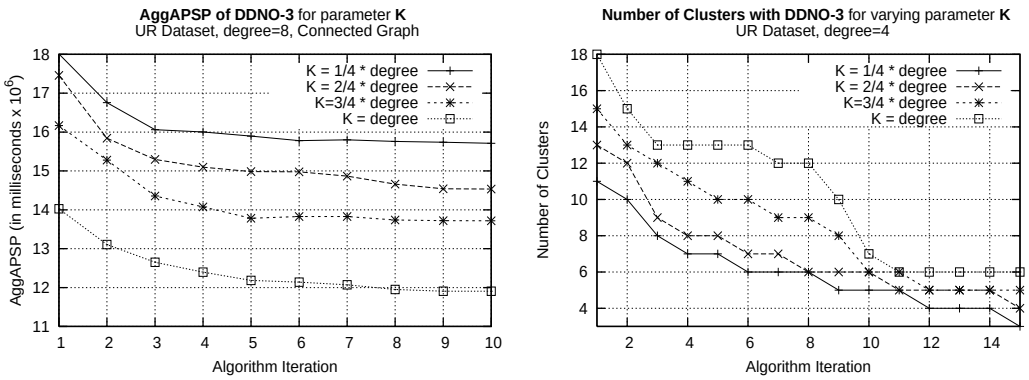


Fig. 8. **Sibling Factor Evaluation** of the DDNO-3 algorithm (graph degree=8) using the UR dataset.

AggAPSP is defined as the sum of shortest path latencies between all vertices in the overlay graph. If the shortest path between any two vertices is indicated as $\infty$ (i.e. these two vertices are not connected), then AggAPSP will incorrectly return $\infty$. Therefore in the next experiment we used as our measure the number of disconnected network segments (clusters).

Figure 8 (right), shows that a larger sibling factor K significantly increases the number of clusters in the overlay topology. Additionally, we observe that a larger sibling factor also delays the stabilization process of the DDNO algorithm. Note that the DDNO algorithm will only stabilize after $K$ siblings nodes have been found. For any other number below this threshold, a node will continue to seek for other siblings in the network. However when the sibling factor is very large, then this has similarly to the Greedy Short algorithm, a negative effect on the overlay performance as it finally results in many disconnected clusters of topologically close-by nodes.

While it is hard to define the optimal parameter K, as this is largely depends on the network instance, we believe that the selection of *degree/2*, presents a good tradeoff between overlay performance and connectivity in networks of arbitrary degree.

### *5.4 Overhead Evaluation*

In order to assess the overhead of the DDNO technique, we investigate the average number of hops each lookupDN message $\ell$ traverses before finding some sibling node. These results are obtained, as with the previous subsection, from the execution using the UR dataset. As we can see in figure 9 (left), $\ell$ initially requires about eight messages (hops), before it is able to locate its siblings. In the subsequent iterations the various ZoneCaches get populated, which consequently lead more $\ell$ messages to the right regions. The plot indicates that after

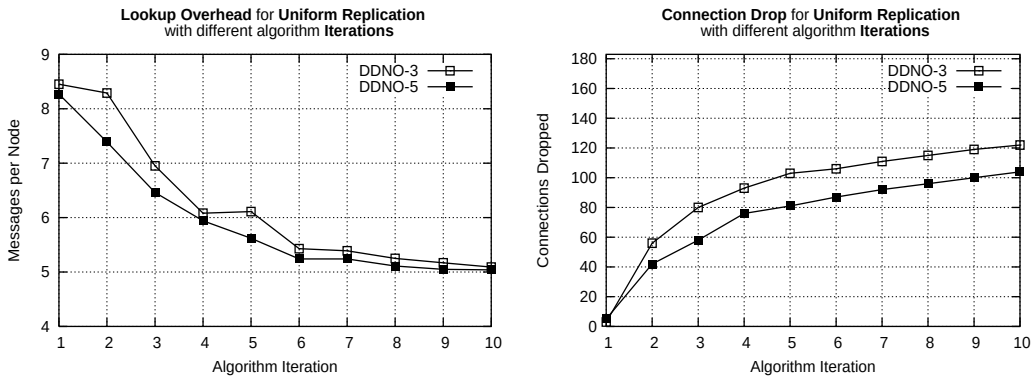


Fig. 9. **Overhead Evaluation** of the DDNO-3 (left) and DDNO-5 (right) techniques using the UR dataset.

the sixth iteration, $\ell$ requires only five hops for both DDNO-3 and DDNO-5 although DDNO-5 stabilizes slightly faster because of its expanded coverage.

The second overhead parameter that we investigate is the total number of temporary connections that are swapped with sibling connections once the latter are found. Figure 9 (right) indicates that DDNO-5 is again able to perform slightly better because $\ell$ messages are resolved faster, which consequently eliminates the need for temporary connections. We can further see that the total number of swapped connections for DDNO-5 and DDNO-3 is 100 and 120 respectively. This accounts to only a drop of $\approx$7% of the total connections in the case of DDNO-5 and $\approx$8.5% in the case of DDNO-3.

### *5.5 Dynamic Environment Evaluation*

Network failures in overlay systems are commonplace because of the misusage exhibited at the application layer (e.g. users shut down their PCs without disconnecting) and the overwhelming amount of generated network traffic. Such failures generate a dynamic environment in which peers are leaving or joining the network in an ad-hoc manner. A highly dynamic environment neutralizes the purpose of the ZoneCaches, as cached information might become outdated before it gets the chance to be utilized.

We choose to evaluate only *DDNO-3*, where each node uses a ZoneCache with a 3-hop radius, since our preliminary runs on networks of different sizes, indicated that such a setting consistently offered good performance at a low overhead. In order to simulate network failures, we disconnect at each *iteration* a fraction of nodes. The failure rates we used are $\{0\%, 5\%, 10\%, 20\%\}$.

In figure 10 (left) we plot the number of resolved lookupDN messages after running DDNO-3 using the RR dataset. The figure indicates that $\approx$89% and

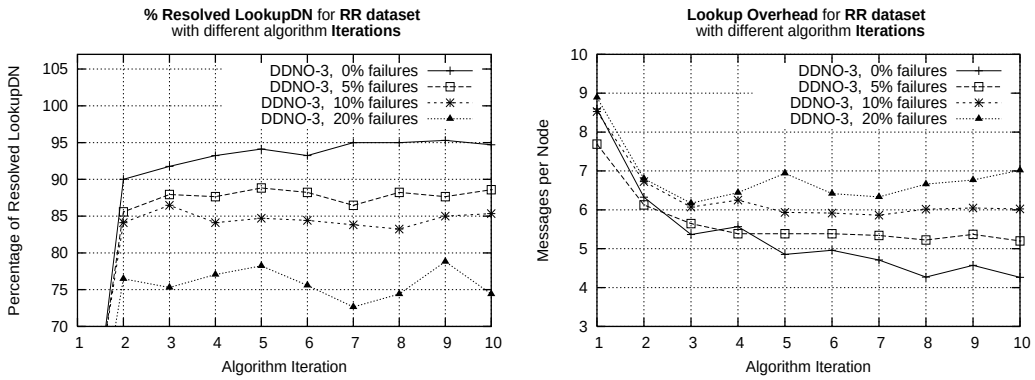


Fig. 10. **Dynamic Environment** Evaluation of the DDNO-3 algorithm over the RR dataset, in a dynamic network topology where nodes leave and arrive.

≈85% of the messages are resolved at 5% and 10% of failures respectively. Therefore low degrees of node failures don't significantly affect the performance of our scheme. With 20% failures the number of resolved lookup messages drops to 75%. Although this might be acceptable in some settings, the fact that the number of hops required by each message increases over time (see figure 10 (right)), might make our scheme not appropriate in such a dynamic setting. On the same figure 10 (right), we can also see that with 5% and 10% of failures the number of hops required by each messages stabilizes at 5 and 6 hops respectively. It is important to remind that in DDNO there is no explicit mechanism to delete outdated entries in the distributed ZoneCaches as this would introduce an additional messaging cost. Each node therefore relies on its LRU policy to invalidate old entries.

### *5.6 Scalability Evaluation*

In this subsection we show how our technique scales to larger network sizes by measuring the percentage of resolved lookupDN messages and the average number of hops each message travels. More specifically, we utilize the LRR-5K and LRR-10K datasets which were described in section 4.2. These datasets consist of 5,000 and 10,000 nodes respectively. We used the *DDNO-3* topology, in which each node has a caching radius of three and node $degree = 12$.

In figure 11 (left) we can see that in the first iteration approximately 57% and 60% of the lookupDN messages are resolved for the LRR-5K and LRR-10K datasets respectively. This low rate is attributed to the fact that the various ZoneCaches are not populated adequately. In the subsection iterations however, the lookupDN procedure is able to resolve ≈95-98% of the requests. In figure 11 (right), we can see that after the first two iterations, lookupDN messages are resolved within 4-5 hops. This result shows that resolving lookup in a completely decentralized fashion doesn't actually impose a large overhead

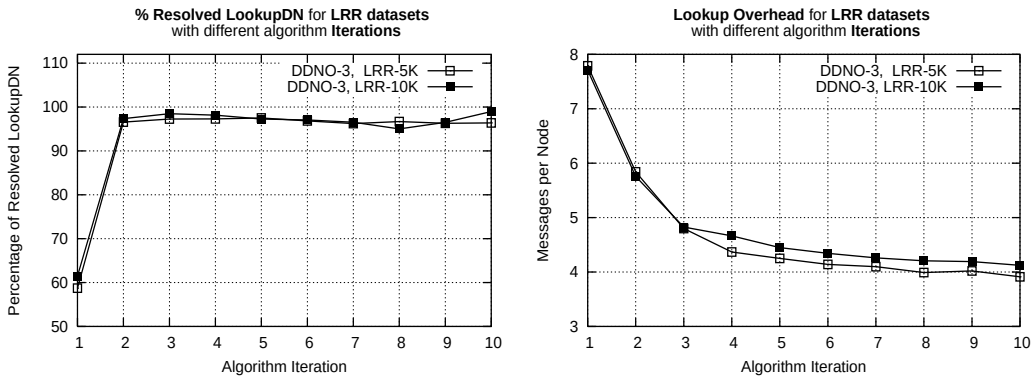


Fig. 11. **Scalability Evaluation** of the DDNO-3 and DDNO-5 algorithms over the LRR-5K and LRR-10K datasets which are networks of 5,000 and 10,000 nodes respectively.

of messages even in larger topologies. Another interesting observation is that although the network size was doubled in the LRR-10K dataset, the number of hops taken by the lookupDN message has only slightly increased (≈0.2).

## 6 Related Work

The need of topologically-aware *unstructured* overlay networks has been addressed in [23]. In the proposed BinSL algorithm [23], which was evaluated in this work, end-to-end delays are minimized using a system of $k$ landmarks. Recently an approach to create resilient unstructured overlays with small diameters was proposed in [28]. In the proposed algorithm a node selects from a set of $k$ nodes, $r$ nodes at random ($r \subset k$) and then finds from the rest $f=k$-$r$ nodes the ones that have the largest degree. The algorithm results in networks with power-law distributions of node degrees differentiating it therefore from DDNO in which we have a uniform distribution.

In [18], the authors propose the *Location-aware Topology Matching (LTM)* technique for unstructured overlay networks. In the proposed scheme, peers use the network delay information between two nodes as a metric for measuring the cost between the nodes and dynamically choose to connect to physically closer nodes and disconnect from distant ones. The network delay between two nodes is computed dynamically by periodically sending special TTL2-detector flooding messages with an initial TTL value of 2. With these messages, a peer can compute the cost of the paths to a source peer.

While *LTM* has the same objectives with DDNO, our scheme differs from the above in the following important ways: (i) We use domain names to identify physically close peers, instead of using a measurement-based approach. Thus, our scheme has the advantage that it does not depend on the current value

of the measurement. For example, in [18], the amount and variation of traffic between the nodes affects the creation and deletion of the connections. If there is heavy traffic, some connections may be town down; affecting the stability of their scheme. (ii) Although domain name is a global metric, our approach establishes peer connections within (and across) the domains, with respect to this global metric. (iii) Our scheme does not depend on the clock synchronization of the peers. In [18], such synchronization is necessary to compute the delays between the nodes. The synchronization, however, might be an expensive process as it requires the exchange of messages among nodes.

In the Vivaldi [7] algorithm, nodes are assigned synthetic coordinates so that the Euclidean distance between them estimates the actual network latency. However, the algorithm requires re-computation of the coordinates on an ongoing basis as opposed to DDNO in which sibling nodes are located only during initialization.

Topologically-aware overlays have also been addressed in the context of *Structured* P2P overlays in [3,23,29,33]. These systems rely on some hashing scheme which allows nodes to quickly send messages to some destination node. Although structured overlays are of particular importance in applications such as decentralized web caches [12], they are not appropriate for content-based retrieval systems [32] and large-scale systems with transient user populations [4]. Li *et al* [17] propose techniques to construct overlay networks (meshes). However, their techniques are not distributed.

The *Cluster-based Architecture for P2P (CAP)*, proposed by Krishnamurthy et. al in [16], shares many similarities in its objective with *DDNO*. The authors propose an architecture in which topologically close-by nodes, identified by their IP addresses, are clustered together using a centralized *clustering service*. Each cluster then features a *cluster delegate* node that acts as a content directory service and which is utilized for efficiently answering queries. Rather than flooding the network, nodes can directly interact with the delegate in order to obtain an answer to their query. DDNO has two subtle differences with CAP: i) We propose the deployment of distributed lookup queries to locate the cluster to which nodes should join rather than utilizing a centralized clustering service, ii) We utilize domain names in order to identify close-by nodes instead of IP addresses. Although IPs have been used in the past to determine network proximity [21], we believe that the current structure of most major Autonomous Systems, in which nodes under the same administrative control might have different CIDR prefixes, would significantly degrade the clustering efficiency.

Similarly to CAP, network-awareness is also addressed in the context of large-scale service overlays [14]. In the proposed scheme, a hierarchically fully connected topology of nodes that are clustered based on their distances is con-

structed. Although the centralized clustering component might be fast and accurate, decentralized approaches are more scalable and less vulnerable to denial of service attacks and censorship [20].

Application-layer multicast systems such as Narada [5] initially construct a richer connected graph (mesh) and then use some optimization algorithm to generate a mesh that has certain performance properties. As part of the mesh quality improvement algorithm, Narada nodes randomly probe each other and calculate the perceived gain in utility. We believe that our approach is simpler and cheaper in terms of messages. It is furthermore designated for larger groups of members, which might leave and join in an ad-hoc manner.

Finally, on an alternative approach, recent work [34] proposes to organize the nodes in semantic groups. Given a query, the question is how to locate the most relevant semantic groups and then flood the query within the group. Efficiently classifying content in groups is an expensive procedure in practice, as it requires the continuous replication of content summaries between neighboring nodes . Finally, these systems do not take into account the underlying network characteristics making it inappropriate for systems that rely on wide-area packet routing.

## 7 Conclusions & Future Work

In this work we propose and evaluate *DDNO (Distributed Domain Name Order)*, which is a distributed technique to make unstructured overlays topologically aware. We compare DDNO with a number of other overlay construction techniques in both centralized and distributed settings. Our experiments indicate that DDNO is an attractive technique for topologically aware overlay construction as it optimizes many desirable properties such as end-to-end delays, diameter and avoids network partitioning, scales to large overlay networks and works well in dynamic environments. We believe that our technique is simple which will enable seamless integration into existing overlay systems with minimum changes to the respective protocols. In the future we want deploy our middleware platform, which is currently under development, over the PlanetLab [22] distributed overlay testbed which is expected to run over 1000 geographically distributed machines in the next few years.

**Acknowledgements:** We would like to thank Dimitrios Gunopulos (UCR), Neal Young (UCR and Akamai), Arthur W. Berger (Akamai and MIT) and Sylvia Ratnasamy (Intel) for the constructive discussions, ideas and suggestions.

## References


[1] Batagelj V. and Mrvar A. "PAJEK - Program for large network analysis", Connections, 21:47–57, 1998.

[2] Bollobás B. "Modern Graph Theory, Graduate Texts in Mathematics" vol. 184, Springer-Verlag, New York, 1998.

[3] Castro M., Druschel P., Charlie Hu Y., Rowstron A. "Topology-aware routing in structured peer-to-peer overlay networks", *In IFIP/ACM Middleware*, 2001.

[4] Chawathe Y., Ratnasamy S., Breslau L., Lanham N., Shenker S., "Making Gnutella-like P2P Systems Scalable", In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, Pages 407-418, 2003.

[5] Chu Y-H, Rao S.G., Zhang H. "A Case For End System Multicast", In *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, St. Clara, CA, pp. 1-12, 2000.

[6] Crespo A. and Garcia-Molina H. "Routing Indices For Peer-to-Peer Systems", In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Vienna, Austria, Page 23, 2002.

[7] Dabek F, Cox R, Kaashoek F, Morris R, "Vivaldi: A Decentralized Network Coordinate System", In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, Portland, OR, Pages 15-26, 2004.

[8] Garey M.R. and Johnson D.S. "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman, 1979.

[9] Gnutella, `http://www.gnutelliums.com/`.

[10] Gross J.L. and Yellen J. "Graph theory and its applications", CRC, 1999.

[11] Hansen, T., Otero, J., McGregor, A., Braun, H-W., "Active measurement data analysis techniques", In *Proceedings of the International Conference on Communications in Computing*, Las Vegas, Nevada, Page 105, 2000.

[12] Iyer S., Rowstron A., Druschel P., "SQUIRREL: A decentralized, peer-to-peer web cache", In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, Monterey, CA, USA, Pages 213-222, 2002.

[13] Kalogeraki V., Gunopulos D., and Zeinalipour-Yazti D. "A Local Search Mechanism for Peer-to-Peer Networks", *In ACM CIKM'02*, McLean, Virginia USA, November 2002.

[14] Jin J. and Nahrstedt K., "Large-Scale Service Overlay Networking with Distance-Based Clustering", In Proc. of ACM/IFIP/USENIX International Middleware Conference (Middleware 2003), Rio de Janeiro, Brazil, June, 2003

[15] Kazaa, `http://www.kazaa.com/`

[16] Krishnamurthy B., Wang J. and Xie Y., "Early Measurements of a Cluster-based Architecture for P2P Systems", In Internet Measurement Workshop San Francisco, CA, USA, 2001.

[17] Li Z. and Mohapatra P. "The impact of topology on overlay routing service", In *The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, Hong Kong, China, 2004.

[18] Liu Y., Liu X., Xiao L., Ni L. M., Zhang X. "Location-aware topology matching in P2P systems", In *The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, Hong Kong, China, 2004.

[19] Mockapetris P. "Domain Names - Implementation and Specification", RFC-1035, Network Working Group, Nov. 1987.

[20] Napster, `http://www.napster.com/`.

[21] Padmanabhan V.N. and Subramanian L., "An investigation of geographic mapping techniques for internet hosts", In Proc. of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications San Diego, CA, Pages: 173-185, 2001

[22] PlanetLab `http://www.planet-lab.org/`.

[23] Ratnasamy S., Handley M., Karp R., Shenker S. "Topologically-Aware Overlay Construction and Server Selection", In *The 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, USA, 2002.

[24] Ratnasamy S., Francis P., Handley M., Karp R., Shenker S. "A Scalable Content-Addressable Network", In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, San Diego, CA, USA, Pages 161-172, 2001.

[25] Ripeanu M, Foster I. and Iamnitchi A., "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design", In IEEE Internet Computing Journal, vol. 6(1) 2002.

[26] Rowstron A. and Druschel P., "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", In *IFIP/ACM International Conference on Distributed Systems Platforms* Heidelberg, Germany, 2001.

[27] Stoica I., Morris R., Karger D., Kaashoek M.F., Balakrishnan H. "Chord: A scalable peer-to-peer lookup service for Internet applications". In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, San Diego, CA, pp. 149-160, 2001.

[28] Wouhaybi R. and Campbell A. "Phenix: Supporting Resilient Low-Diameter Peer-to-Peer Topologies", In *The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, Hong Kong, China, 2004.

[29] Xu Z., Tang C., Zhang Z. "Building Topology-Aware Overlays using Global Soft-State", In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, Providence, RI, USA, Page 500, 2003.

[30] Yang B., and Garcia-Molina H. "Efficient Search in Peer-to-Peer Networks". In *Proceedings of the 22nd International Conference on. Distributed Computing Systems* Vienna, Austria, Pages 5-14, 2002.

[31] Zeinalipour-Yazti D. and Folias T., "Quantitative Analysis of the Gnutella Network Traffic", Tech. Rep. UC-CS-89, UCR.

[32] Zeinalipour-Yazti D., Kalogeraki V., Gunopulos D. "Exploiting Locality for Scalable Information Retrieval in Peer-to-Peer Systems", Information Systems Journal, Elsevier Publications, Volume 30, Issue 4, Pages 277-298, 2005.

[33] Zhao B.Y., Duan Y., Huang L., Joseph A.D., Kubiatowicz J.D. "Brocade: landmark routing on overlay networks", In *First International Workshop on Peer-to-Peer Systems*, Cambridge MA, LNCS Vol. 2429, Pages 34-44, 2002.

[34] Zhu Y., Yang X., Hu Y., "Making Search Efficient on Gnutella-like P2P Systems", *In 19th International Parallel and Distributed Processing Symposium*, Denver, CO, USA.