Topology-Centric Look-Up Service

L. Garcés-Erice[1], K.W. Ross[2], E.W. Biersack[1], P.A. Felber[1], and G. Urvoy-Keller[1]

[1] Institut EURECOM
06904 Sophia Antipolis, France
{garces|erbi|felber|urvoy}@eurecom.fr

[2] Polytechnic University
Brooklyn, NY 11201, USA
ross@poly.edu

Abstract. Topological considerations are of paramount importance in the design of a P2P lookup service. We present TOPLUS, a lookup service for structured peer-to-peer networks that is based on the hierarchical grouping of peers according to network IP prefixes. TOPLUS is fully distributed and symmetric, in the sense that all nodes have the same role. Packets are routed to their destination along a path that mimics the router-level shortest-path, thereby providing a small "stretch". Experimental evaluation confirms that a lookup in TOPLUS takes time comparable to that of IP routing.

1 Introduction

Several important proposals have recently been put forth for providing a distributed peer-to-peer (P2P) lookup service, including Chord [1], CAN [2], Pastry [3] and Tapestry [4]. These lookup services can be compared in many ways, including speed of lookup, implementation complexity, symmetry, potential for caching, and resilience to faults and attacks. It turns out that for many measures — like speed of lookup and potential for caching — it is highly desirable that the lookup service takes the underlying IP-level topological considerations into account. Acknowledging the importance of topological considerations, researchers have recently proposed a number of modifications to the original lookup services that take topology into special consideration [5–7].

It is our position that topological considerations are of paramount importance in a P2P lookup service, and therefore, when designing a lookup service, topology needs to be built in from the ground up. In this paper we explore the following issues:

1. How can we design a P2P lookup service for which topological considerations take precedence?
2. What are the advantages and disadvantages of such a topology-centric design?
3. How can the topology-centric design be modified so that the advantages of the original design are preserved but the disadvantages are abated?

To respond to the first question, we propose a new lookup service, **Topology-Centric Look-Up Service (TOPLUS)**, that has been expressly designed from the ground up to exploit the topological structure of the underlying Internet. In TOPLUS, nodes that are topologically close are organized into groups. Furthermore, groups that are topologically close are organized into supergroups, and supergroups that are topologically close are organized into hypergroups, etc. The groups within each level of the hierarchy can be heterogeneous in size, and the fan-outs from the groups can also be heterogeneous. The groups can be derived directly from the network prefixes contained in BGP tables or from other sources. TOPLUS has many strengths, including:

– **Stretch:** Packets are routed to their destination along a path that mimics the router-level shortest-path distance, thereby providing a small "stretch".
– **Caching:** On-demand P2P caching of data is straightforward to implement, and can dramatically reduce average file transfer delays.
– **Efficient forwarding:** As we shall see, nodes can use highly-optimized IP longest-prefix matching techniques to efficiently forward messages to the next hop.
– **Symmetric:** Although TOPLUS has been carefully designed to reflect the underlying network topology, all nodes have similar responsibilities.

TOPLUS is an "extremist's design" to a topology-centric lookup service. At the very least, it serves as a benchmark against which other lookup services can compare their stretch and caching performance.

After describing the TOPLUS lookup service in detail, we evaluate its performance using several group structures derived from a large set of prefixes, obtained mainly from BGP routing tables. Experimental results show that TOPLUS provides a small stretch in most configurations. To obtain near-optimal stretch figures, however, nodes must maintain quite large routing tables.

This paper is organized as follows. We present related work at the end of this section. In Section 2 we describe the TOPLUS design, and we elaborate on its limitations and possible solutions in Section 3. In Section 4 we describe how we obtained the nested group structures from BGP tables and our measurement procedure for evaluating the average stretch. We then provide and discuss our experimental results. We conclude in Section 5.

Related Work

In [5], the authors show how the original CAN design can be modified to account for topological considerations. Their approach is to use online measurement techniques to group nodes into "bins". Although this measurement-based, binning technique can significantly reduce CAN's stretch, the resulting stretch remains significant in their simulation results.

In [6], the authors examine the topological properties of a modified version of Pastry. In this design, a message typically takes small topological steps initially and big steps at the end of the route. We shall see that TOPLUS does the opposite, initially taking a large step, then a series of very small steps. Although [6] reports significantly lower stretches than other lookup services, it still reports an average stretch of 2.2 when the Mercator [8] topology model is used. Coral [9] has been recently proposed to adapt Chord to the Internet topology. Coral organizes peers in clusters and uses a hierarchical lookup of keys that tries to follow a path inside one peer's cluster whenever possible. The query is passed to higher-level clusters when the lookup can't continue inside the original cluster.

Cluster-based Architecture for P2P (CAP) [10] is a P2P architecture that has been built from the ground up with topological considerations. However, TOPLUS differs from CAP in many ways. Most importantly, CAP is an unstructured P2P architecture whereas TOPLUS is a structured DHT-based architecture. Also, CAP uses a two-level hierarchy whereas TOPLUS uses a multi-level hierarchy, and CAP uses "supernodes" for managing groups whereas TOPLUS uses a symmetric design. Nevertheless, although TOPLUS does not mandate a specific clustering technique, we believe the clustering procedures of Krishnamurthy and Wang [11, 12] used in CAP are currently among the most promising techniques to create the groups in TOPLUS.

2 Overview of TOPLUS

In a P2P lookup service, each key is under the responsibility of some up node. Given a message containing key k, the P2P lookup service routes the message to the current up node that is responsible for k. The message travels from source node n_s, through a series of intermediate peer nodes $n_1, n_2, \ldots, n_v$, and finally to the destination node, n_d, which is the node responsible for k.

The principal goals of TOPLUS are as follows: (1) Given a message with key k, source node n_s sends the message (through IP-level routers) to a first-hop node n_1 that is "topologically close" to n_d; (2) After arriving at n_1, the message remains topologically close to n_d as it is routed closer and closer to n_d through the subsequent intermediate nodes, until it finally reaches n_d. Clearly, if the lookup service satisfies these two goals, the stretch should be very close to 1. We now formally describe TOPLUS in the context of IPv4.

Let I be the set of all 32-bit IP addresses.[3] Let $\mathcal{G}$ be a collection of sets such that $G \subseteq I$ for each $G \in \mathcal{G}$. Thus, each set $G \in \mathcal{G}$ is a set of IP addresses. We refer to each such set G as a **group**. Any group $G \in \mathcal{G}$ that does not contain another group in $\mathcal{G}$ is said to be an **inner group**. We say that the collection $\mathcal{G}$ is a **proper nesting** if it satisfies all the following properties:

1. $I \in \mathcal{G}$.
2. For any pair of groups in $\mathcal{G}$, the two groups are either disjoint, or one group is a proper subset of the other.
3. For each $G \in \mathcal{G}$, if G is not an inner group, then G is the union of a finite number of sets in $\mathcal{G}$.
4. Each $G \in \mathcal{G}$ consists of a set of contiguous IP addresses that can be represented by an IP prefix of the form $w.x.y.z/n$ (for example, 123.13.78.0/23).

As shown in Section 4, the collection of sets $\mathcal{G}$ can be created by collecting the IP prefix networks from BGP tables and/or other sources [11, 12]. In this case, many of the sets $\mathcal{G}$ would correspond to ASes, other sets would be subnets in ASes, and yet other sets would be aggregations of ASes. This approach of defining $\mathcal{G}$ from BGP tables require that sets be massaged so that the four properties of a proper nesting are satisfied. Additionally, some of the groups may be aggregated, and new artificial tiers may be introduced, in order to reduce the size of the nodal routing tables. Note that the groups differ in size, and that the number of groups within a group (the fanout) is also different from group to group.

[3] For simplicity, we assume that all IP addresses are permitted. Of course, some blocks of IP addressed are private and other blocks have not been defined. TOPLUS can be refined accordingly.

It is straightforward to see that if $\mathcal{G}$ is a proper nesting, then the relation $G \subset G'$ defines a partial ordering over the sets in $\mathcal{G}$. This partial ordering defines a partial-order tree with multiple tiers. The set I is at tier-0, the highest tier. A group G belongs to tier 1 if there does not exist a G' (other than I) such that $G \subset G'$. We define the remaining tiers recursively in the same manner (see Figure 1). Note that each of the leaf groups in the partial order tree is an inner group.

2.1 Node State

Each up node is required to know the IP addresses of a relatively small part of all up nodes. To describe this node state, let L denote the number of tiers in the tree, let U be the set of all current up nodes and consider a node $n \in U$. Node n is contained in a collection of telescoping sets in $\mathcal{G}$; denote these sets by $H_N(n), H_{N-1}(n), \cdots, H_0(n) = I$, where $H_N(n) \subset H_{N-1}(n) \subset \cdots \subset H_0(n)$ and $N \leq L$ is the tier depth of n's inner group. Except for $H_0(n)$, each of these telescoping sets has one or more siblings in the partial-order tree (see Figure 1). Let $\mathcal{S}_i(n)$ be the set of siblings groups of $H_i(n)$ at tier i. Finally, let $\mathcal{S}(n)$ be the union of the sibling sets $\mathcal{S}_1(n), \cdots, \mathcal{S}_N(n)$.

As part of node n's state information, for each group $G \in \mathcal{S}(n)$, node n should know the IP address of at least one node in G. Also, node n should know the IP addresses of all the other nodes in its inner group. We refer to the collection of these two sets of IP addresses as node n's **routing table**, which constitutes node n's state.

Now consider the number of IP addresses stored in a node's routing table. Assume that the node's inner group is in the lowest tier, L. Then, the total number of IP addresses in the node's routing table is $|H_L(n)| + |\mathcal{S}(n)|$. During the formation of the nested groups, it is desirable to prevent the routing table from becoming too large in order to minimize nodal storage and computation. At the same time, is also desirable to prevent a node's inner-group table from becoming too large.

2.2 XOR Metric

Each key k' is required to be an element of I', where I' is the set of all s-bit binary strings, where s is fixed and $s \geq 32$. A key can be drawn uniformly randomly from I', or it can be biased as we will describe later. For a given key $k' \in I'$, denote k for the 32-bit suffix of k'. Note that k is in I and hence corresponds to an IP address. Throughout the discussion below, we will typically refer to a key's 32-bit suffix rather than to the original key itself. Thus any key or IP address k can be expressed as $k = k_{31}k_{30}\ldots k_1k_0$.

The XOR metric defines the distance between two ids j and k as:

$$d(j,k) = \sum_{\nu=0}^{31} |j_\nu - k_\nu| \cdot 2^\nu$$

The metric $d(j,k)$ has the following properties:

– If $d(i,k) = d(j,k)$ for any k, then $i = j$. This is a curious property, which does not hold for standard metrics in R^n.
– $\max d(j,k) \leq 2^{32} - 1$.
– Let $p(j,k)$ be the number of bits in the common prefix of j and k. If $p(j,k) = m$, $d(j,k) \leq 2^{32-m} - 1$.
– If $d(i,k) \leq d(j,k)$, then $p(i,k) \geq p(j,k)$.

Note that $d(j,k)$ is a refinement of longest-prefix matching. If j is the unique longest-prefix match with k, then j is the closest to k in terms of the metric. However, if there are ties in the longest-prefix match, the metric will break the ties. The Kademlia DHT [13] also uses the XOR metric.

For any given key k, let n^* be the node in U that minimizes $d(k,n)$, $n \in U$. Node n^* is said to be "responsible" for k. Because two nodes cannot be equally close to k, the responsible node x^* is uniquely defined.

2.3 The Lookup Algorithm

Each node n has a lookup API. Node n_s inputs into the API a key k (that is, the 32-bit suffix of k'), and the API returns the up node n_d that, among all the up nodes, is the closest to k in terms of the XOR metric.

Suppose node n_s wants to look up k. Node n_s determines the node in its routing table that is closest to k in terms of the XOR metric. Suppose node n_j is the closest. Then node n_s forwards the message to n_j. The process continues, forwarding the message from peer to peer, until the message with key k reaches a node n_d such that the closest node to k in n_d's routing table is n_d itself. It is trivial to prove that n_d is the node that is responsible for k.

Because the groups and their nestings have been derived directly from the underlying topology, TOPLUS is topology centric. If the set of groups form a proper nesting, then it is straightforward to show that the number of hops in a look up is at most $L+1$, where L is the depth of the partial-order tree. Typically, big topological jumps will be made initially (for example, going

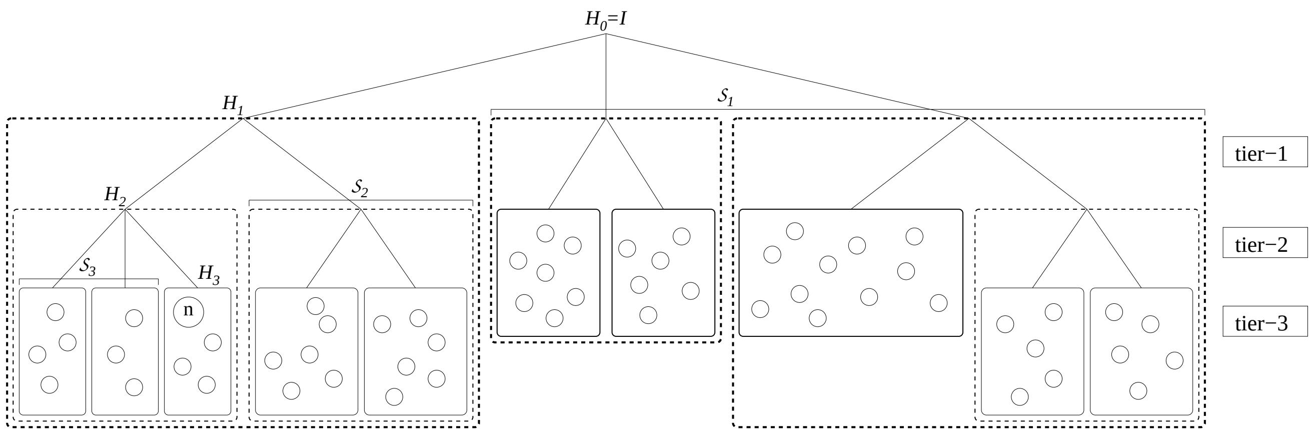

Fig. 1. A sample TOPLUS hierarchy (inner groups are represented by plain boxes).

from source AS to destination AS); subsequent jumps will typically be decreasingly shorter as the message converges on the target subnet. Note that TOPLUS satisfies the two goals described at the beginning of the section. In the first hop the message will be sent to a node n_1 that is in the same group, say G, as n_d. The node n_1 will likely be topologically close to n_d. Once the message arrives at G, it will remain in G until it arrives at n_d.

As previously mentioned, each node in TOPLUS mimics a router in the sense that it routes messages based on a generalization of longest-prefix matching of IP addresses. To this end, nodes can use highly-optimized longest-prefix matching schemes [14] deployed in high-speed routers for routing messages.

2.4 Overlay Maintenance

When a new node n joins the system, n asks an arbitrary existing node to determine (using TOPLUS) the closest node to n (using n's IP address as the key). Denote this closest node by n'. Node n then initializes its routing table with n''s routing table. Node n's routing table should then be modified to satisfy a "diversity" property: for each node n_i in the routing table, n asks n_i for a random node in n_i's group. This way, for every two nodes in a group G, their respective sets of delegates for another group G' will be disjoint (with high probability). This "diversity" property aims at making routing more robust, because in case one delegate fails, it is possible to use another node's delegate.

Also, a small fraction of existing node tables should be modified when n joins: all nodes in its group must update their inner group tables.

Maintenance of the overlay network is relatively simple. Note that groups, which are virtual, do not fail; only nodes can fail. Existing groups can be partitioned or aggregated on a slow time scale, as need be. When needed, keys can be moved from one group to another lazily: when a node receives a query for a key that it is not storing, the node can perform itself a query excluding its own group. Once the key is retrieved from its formerly corresponding location, subsequent queries can be normally satisfied.

2.5 On-Demand P2P Caching

An ISP (such as a university, a corporate campus, or a residential ISP) often deploys a Web cache to improve file transfer times. In a similar manner, TOPLUS can provide a powerful caching service.

Suppose that a node n_s wants to obtain the file f associated with key k. Once n_s learns from the lookup service that another up node n_d is responsible for a key k, node n_s asks n_d to send it the file directly. Unfortunately, there may be bottleneck physical links (for example, ISP peering interfaces) between n_s and n_d. It would then be preferable if n_s could obtain a cached copy of file f from a topologically close node, perhaps from a node on the same high-speed LAN as n_s.

To this end, suppose that some group $G \in \mathcal{G}$ wants to provide a caching service to the nodes in G (G could be an inner group, or it could be a group higher up in the hierarchy). Further suppose all pairs of nodes in G can send files to each other relatively quickly. For example, all the nodes in G may be interconnected with a high-speed LAN.

In this distributed caching service, all the nodes in G are configured to first contact the "cache" in G before attempting to download the desired file from the global lookup service. This is done as follows. Let the network prefix for G be denoted by w.x.y.z/r. Now suppose some node $n_s \in G$ wants to find the file f associated with key $k \in I$. Then n_s creates a new key, k_G, which is k but with the first r bits of k replaced with the first r bits of w.x.y.z/r. Node n_s then inserts a message with key k_G into TOPLUS. The lookup service will return to n_s the node n_G that is responsible for k_G. Node n_G will be in G, and all the messages traveling from n_s to n_G will be confined to G. n_s then asks n_G for f. If n_G has f (cache hit), then n_G will send f to n_s at a relatively high rate. If n_G does not have f (cache miss), n_G will use TOPLUS to obtain f from the global lookup service. After obtaining f, n_G will cache f in its local shared storage and pass a copy of f to n_s.

Thus all the nodes in G cooperate to provide a distributed cache with storage aggregated across all the nodes in G. Each node would employ a file replacement policy, such as least recently used. As with an ordinary Web cache, the more popular files are more likely to be cached in G. The techniques in [15] can be used to optimally replicate files throughout G to handle intermittent nodal connectivity. Also, this distributed caching idea can be extended to distributed cache hierarchies (analogous to Web cache hierarchies). Finally, files can be pushed into groups, creating distributed set of CDN nodes in each of the designated groups.

3 Drawbacks and Solutions

Because the TOPLUS design gives precedence to topological considerations, TOPLUS should exhibit excellent stretch and caching performance. But admittedly, these features come by sacrificing other desirable properties in a P2P lookup service. We now discuss some of the drawbacks of the TOPLUS design. As well as approaches to address these drawbacks.

Non-uniform population of id space: The number of keys assigned to an inner group will be approximately proportional to the number of IP addresses covered by the inner group. However, the number of active nodes in an inner group is not necessarily proportional to its size (in terms of IP address coverage). This means that some nodes may be responsible for a disproportionate number of keys.

Lack of virtual nodes: Because nodes have different storage, processing, and bandwidth, it is desirable to assign larger proportion of keys to more powerful nodes. The CAN, Chord, Pastry and Tapestry lookup services can handle heterogeneous nodes by assigning virtual nodes to the more powerful peers. TOPLUS, as currently defined, does not facilitate the creation of virtual nodes.

Correlated node failures: Many applications built on top of a lookup service, including persistent file storage, require that key/data pairs be replicated on multiple nodes. As with Chord, Pastry. and Tapestry, TOPLUS can replicate key/data pairs on successor nodes within the same inner group. However, when replicating in this manner, if an entire inner group fails (for example, if an access link crashes), then all copies of the data for the key become unavailable. Because in the other look up services there is no correlation between node id and locality, these services are not as sensitive to correlated node failures.

We now outline a number of enhancements to TOPLUS that solve or partially solve the problems listed above. The first enhancement is to use a non-uniform distribution when creating keys. Specifically, suppose there are J inner groups, and we estimate the average fraction of active nodes in inner group j to be q_j. Then when assigning a key, we first choose an integer (deterministically) from $\{1, 2, \ldots, J\}$ using the weights $q_1, \ldots, q_J$. Suppose we choose group j, and group j has prefix w.x.y.z/n . We then choose a key uniformly from the set of IP addresses covered by w.x.y.z/n .

In order to address the lack of virtual nodes and other issues resulting from the tight coupling of node ids to IP addresses, we assign each node a permanent "virtual id" uniformly distributed over the address space of the node's inner group. More powerful nodes are assigned multiple permanent virtual ids, thereby creating virtual nodes. In the inner group table, for each IP address in the table we also list all the virtual ids associated with the IP address. After making this change, we modify TOPLUS as follows. As before, the lookup process continues until the message reaches a node n such that the longest prefix match is inside the inner group of n. But once the message is at n, node n now determines, among all the virtual ids in its inner group table, the virtual id that is the closest to the key (according to the exclusive-or metric). n then sends the message to the node corresponding to this virtual id.

Finally, we address the issue of correlated node failures. To solve this problem, when we replicate key/data pairs, we need to distribute the replicas over multiple inner groups. Further, when one inner group fails, we need to detect the failure and move copies of key/data pairs to new inner groups. TOPLUS can be modified to solve this problem, but comprehensive solutions are fairly involved. Due to lack space, we only sketch a partial solution here: rather than using a single hash function, we use K distinct hash functions. A node that publishes some content (data or meta-data) in the system will thus store the content at K different locations that are geographically dispersed with high probability.

4 Benchmarking TOPLUS

In TOPLUS, a group is defined by an IP network prefix. Instead of simply dividing the IP space in arbitrary chunks, we have used IP prefixes obtained from two sources: the BGP routing tables of routers in the Internet, and the prefixes of well-known networks (such as corporate LANs or ISPs). The BGP information offers a coarse-level perspective of the Internet grouping, while IP prefixes from LANs or ISPs provide a finer-level clustering. As shown in [12, 10], the IP prefixes obtained from BGP routing tables form clusters of hosts that are topologically close to each other. Our assumption is that this locality property is preserved to some extent in coarser IP prefixes that regroup clusters (super-clusters), and recursively as IP prefixes become shorter and clusters become larger. Using this IP prefix information, we aim at constructing a topology-centric TOPLUS hierarchy which routes queries to their destinations in a time comparable to that of IP routing.

IP network prefixes were obtained from several sources: BGP tables provided by Oregon University [16] and by the University of Michigan and Merit Network [17]; network IP prefixes from routing registries provided by Castify Networks [18] and RIPE [19]. After merging all this information and eliminating reserved and non-routable prefixes, we have obtained a set of *250,562* distinct IP prefixes that we organize in a partial order tree (denoted as Prefix Tree hereafter). Studies from NLANR [20] show that about 20% of the total usable IP space was routable in 1997, and 25% in 1999. As our tree covers 35% of the total usable IP space, we believe that it offers a nearly complete view of the Internet in 2003.

Source	**IP prefixes provided**
Oregon University	123,593
Michigan U. and Merit Network	104,552
Castify Networks	143,082
RIPE Routing Registry	124,876
Total distinct IP prefixes	*250,562*

Table 1. IP prefixes obtained from different sources.

4.1 Measuring Stretch

The stretch is defined as the ratio between the average latency of TOPLUS routing (using the Prefix Tree) and the average latency of IP routing. Ideally, we could use the `traceroute` [21] tool to measure the delay between arbitrary hosts in the Internet.[4] However, security measures deployed in almost every Internet router nowadays prevent us from using this simple and accurate measurement technique. Therefore, we have used the `King` [22] tool to obtain experimental results. King gives a good approximation of the distance between two arbitrary hosts by measuring the latency between the DNS servers responsible for these hosts.

The general principle of our measurements is shown in Figure 2, where peer at address 1.2.3.4 sends a query for key k, whose responsible peer is 193.56.1.2. Following the TOPLUS routing procedure, peer 1.2.3.4 must first route the query inside the tier-1 group containing k. Peer 1.2.3.4 selects a delegate from its routing table in group 193/8, because $k/32 \subset 193/8$ (note that there cannot be another group G in tier-1 satisfying $k/32 \subset G$). Assuming the delegate is in group 193.50/16, the query is first routed along path labeled with latency d_1 in Figure 2. Then, the delegate selects the (unique) tier-2 group inside 193/8 which contains k: 193.56.0/20. Let the new delegate node be in tier-3 group 193.56.2/24. The query is forwarded to that delegate node along path d_2. Finally, the destination group 193.56.1/24 is reached with the next hop d_3 (we neglect the final forwarding of the query inside the destination group). In contrast to TOPLUS routing, the query would follow path d between 1.2.3.4 and 193.56.1.2 with IP routing. The stretch for this particular case would be $\frac{d_1+d_2+d_3}{d}$.

In general, we consider the length of the path from one peer to another as the weighted average of the length of all possible paths between them. Path weights are derived from the probability of a delegate peer to be in each of the different group at each tier. Assuming a uniform distribution of peers in the Internet, the probability of node n choosing a delegate in group G at tier i with parent $S \in \mathcal{S}_{i+1}(n)$ can be computed as the number of IP addresses in all inner groups descendant of G divided by the number of IP addresses in all inner groups descendant of S. To simplify computations and keep them local to tier i, we approximate this proba-

[4] This can be achieved specifying one of the two host as a gateway packets must pass through.

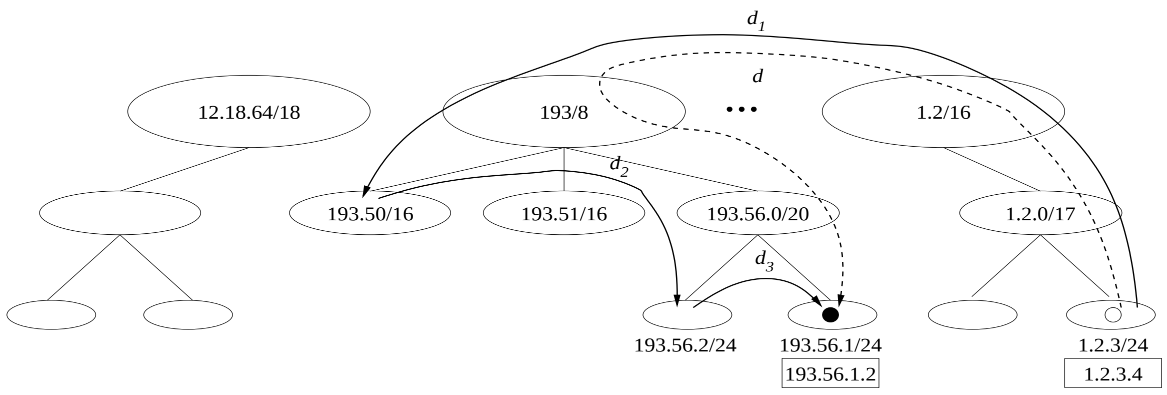

Fig. 2. Path followed by a query in the Prefix Tree.

bility by computing the space of IP addresses covered by the network prefix of G, divided by the space of IP addresses covered by all groups children of S (including G). For instance, for the first hop in Figure 2, the probability of the delegate peer being in group 193.50/16 is $\frac{2^{16}}{2^{16}+2^{16}+(2^{24}+2^{24})} = 0.498$; using the simplified formula, the probability is approximated as $\frac{2^{16}}{2^{16}+2^{16}+2^{12}} = 0.48$.

Consider a query issued by node n_s in inner group S for a key k owned by node n_d in inner group D at tier N. Let $d^T(G, G')$ be the TOPLUS latency between a node in group G and a node in group G' and $d^{IP}(G, G')$ be the corresponding direct IP latency. Let $H_0, \ldots H_N$ (as in Section 2.1) be the telescoping set of groups associated to node n_d (hence $H_0 = I$ and $H_N = D$). To reach n_d, n_s forwards its request to its delegate node n_g belonging to one of the inner groups G in H_1. Hence:

$$E[d^T(S,D)] = \sum_{G \subset H_1} p_G(E[d^{IP}(S,G)] + E[d^T(G,D)]) \tag{1}$$

where p_G is the probability of n_g being in G. Thus, $p_G = \frac{|G|}{|H_1|}$. Note that $E[d^{IP}(S,G)] = 0$ if ever $n_s \in H_1$, since in this case n_s is its own delegate in H_1. The process continues with n_g forwarding the query to its delegate node n'_g in one of the inner groups G' in H_2. The equation for $E[d^T(G,D)]$ is thus similar to Equation (1):

$$E[d^T(G,D)] = \sum_{G' \subset H_2} p_{G'}(E[d^{IP}(G,G')] + E[d^T(G',D)]) \tag{2}$$

where $p_{G'} = \frac{|G'|}{|H_2|}$. N successive recursions thus allow to obtain the value of $E[d^T(S,D)]$.

To obtain the average stretch of the Prefix Tree, we compute the stretch from a fixed origin peer to 1,000 randomly generated destination IP addresses using `King` to measure the delay of each hop. We compute the path length from the origin peer to each destination peer as the average length of all possible paths to the destination, weighted according to individual path probabilities (as described above). Finally, we compute the stretch of the Prefix Tree as the average stretch from the origin peer to each of the 1,000 destination peers. Note that the choice of the origin peer is not important because the routing always starts at tier-1, independent of how deep the origin peer is located in the hierarchy. For the experiments, we chose an origin peer at Institut Eurecom with IP address 193.55.113.1. We used 95% confidence intervals for all measurements.

We now detail the different Prefix Tree configurations that we have considered in our experiments.

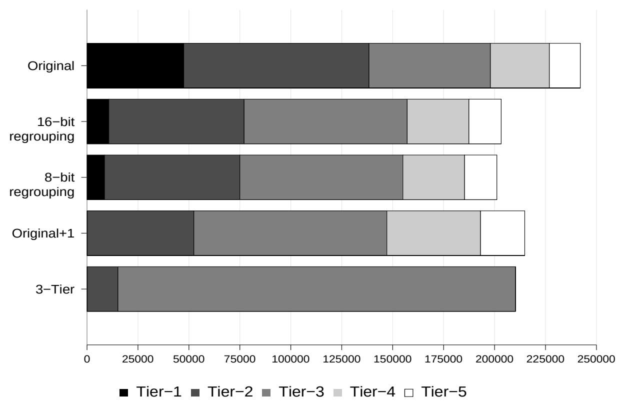

Fig. 3. Number of groups per tier in the Prefix Trees.

Original Prefix Tree. We call "Original Prefix Tree" the tree resulting from the ordering of the IP prefixes using operator $\subset$. The partial order tree has 47,467 different tier-1 groups (prefixes); 10,356 (21.8%) of these groups have at least one tier-2 subgroup; further, 3,206 (30%) of tier-2 groups have at least one tier-3 subgroup. The deepest nesting in the tree comprises 11 tiers. The number of non-inner group at each tier decreases rapidly with the tier depth (following roughly a power-law) and the resulting tree is strongly unbalanced.

Figure 4 (left) shows the distribution of prefix lengths in tier-1. As most prefixes of more that 16 bits do not contain nested groups, the Original Prefix Tree has a large number of tier-1 groups. Consequently, the routing tables of each peer will be large because they have to keep track of one delegate per tier-1 group. On the other hand, since 61% of the IP addresses covered by the tree are within tier-1 inner groups, a large number of peers can be reached with just one hop.

	stretch TOPLUS vs. IP (± confidence interval)		
Tier	Original	16-bit regroup.	8-bit regroup.
1	1.00 (±0.00)	1.00 (±0.00)	1.00 (±0.00)
2	1.29 (±0.15)	1.32 (±0.14)	1.56 (±0.23)
3	1.31 (±0.16)	1.30 (±0.17)	1.53 (±0.23)
4	1.57 (±0.50)	1.41 (±0.20)	1.56 (±0.50)
Mean	1.17 (±0.06)	1.19 (±0.08)	1.28 (±0.09)

Table 2. Stretch obtained in each tree, depending on the tier of the destination peer.

We computed an average stretch of **1.17** for the Original Prefix Tree, that is, a query in TOPLUS takes on average 17% more time to reach its destination than using direct IP routing. In Table 2 we present the average stretch for IP addresses located in inner groups at tiers 1 to 4. As expected, we observe that the deeper we go in the tree to reach a destination, the higher the stretch becomes (because there is a higher probability of making more hops). Note that more than half of the queries had for destination a peer in a tier-2 group.

Modified Prefix Trees. As previously mentioned, a large number of groups are found in tier-1 and all peers in the network must know a delegate in each of those groups. In order to reduce the size of the routing tables, we modify the tree by "aggregating" small groups that have a long prefix into larger groups not present in our IP prefix sources. We consider groups to be "small" if their prefix is longer than 16 bits; this represents 38,966 tier-1 groups for our experimental data.

16-bit regrouping: A first approach consists in aggregating small groups into 16-bit aggregate prefix groups. This means that any tier-1 prefix `a.b.c.d/r` with $r > 16$ is moved to tier 2 and a new 16 bit prefix `a.b/16` is inserted at tier-1. We call this approach "16-bit regrouping". This process creates 2,161 new tier-1 groups, with an average of 18 subgroups in each of them. The distribution of tier-1 prefixes is shown in Figure 4 (middle).

The resulting tree contains 10,709 tier-1 groups, 50% (5,454) of which contain subgroups. We have measured an average stretch of **1.19** for that tree (see Table 2). These results indicate that 16-bit regrouping essentially preserves the low stretch.

8-bit regrouping: We have experimented with a second approach to prefix regrouping, which consists in using coarser, 8-bit aggregate aggregate prefix groups. Any tier-1 prefix `a.b.c.d/r` with $r > 16$ is moved to tier 2 and a new 8 bit prefix `a/8` is inserted at tier-1 (if it does not already exist). We call this approach "8-bit regrouping". This process creates 45 new tier-1 groups, with an average of 866 subgroups in each of them. The distribution of tier-1 prefixes is shown in Figure 4 (right).

The resulting tree contains 8,593 tier-1 groups (more than 5 times less than our Original Prefix Tree). 38% (3,338) of these groups contain subgroups and almost half of tier-2 groups (1,524) have again subgroups. The tree is clearly becoming more balanced and, as a direct consequence of the reduction of tier-1 groups, the size of the routing table in each peer becomes substantially smaller. We have measured an average stretch of **1.28** for the new tree (see Table 2). This remarkable result demonstrates that, even after aggressive aggregation of large sets of tier-1 groups into coarse 8-bit prefixes, the low stretch property of the original tree is preserved.

4.2 Routing Table Size

The principal motivation for prefix regrouping is to reduce the size of the routing tables. We estimate the size of the routing tables by choosing 5,000 random uniformly distributed IP addresses (peers); for each of these peers n, we examine the structure of the tree to determine the sibling sets $\mathcal{S}(n)$ and the inner group nodes $H_N(n)$, and we compute the size $|\mathcal{S}(n)|+|H_N(n)|$ of the routing table of node n.

Table 3 shows the mean routing table size depending on the tier of the peer, as well as the average size

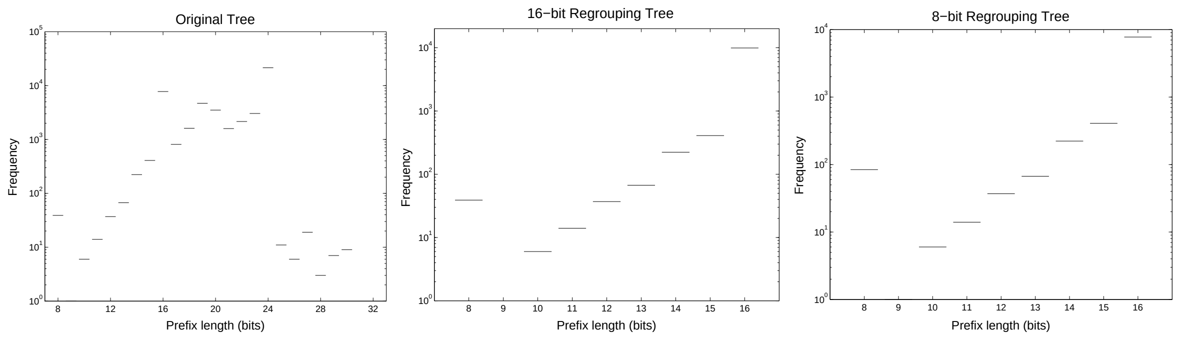

Fig. 4. Prefix length distribution for the tier-1 groups of the three Prefix Trees: Original, 16-bit regrouping and 8 bit regrouping.

over all tiers. Looking at the tree structure, one immediately understands that the route table size is mainly determined by the number of tier-1 groups. If we eliminate their delegates from the routing table, we see that the size of the routing tables needed to route queries *inside* each tier-1 group remains small.

	Mean routing table size					
Tier	Original		16-bit regroup.		8-bit regroup.	
1	47,467	0	10,709	0	8,593	0
2	47,565	98	10,802	93	8,713	120
3	47,654	187	10,862	153	8,821	228
4	47,796	329	11,003	294	8,950	357
5	47,890	423	11,132	423	9,016	423
Mean	47,547	80	10,788	79	8,699	106

Table 3. Mean routing table size in each tree depending on the tier of a peer. For each tree, the left column is the full routing table size and the right column is the size without tier-1 groups.

Even using 8-bit regrouping, routing tables count more than 8,000 entries (see Table 3). We recognize that this is a very large number of peers to be aware of, even when other P2P systems can present similar figures, such as [23], where all peers know each other. To further reduce the routing table sizes, we transform the "original" and the "16-bit-regrouping" trees such that all tier-1 prefixes are 8-bit long, which will limit the number of tier-1 groups to *at most* 256. We refer to the resulting trees as "Original+1" and "16-bit+1". For this purpose, any tier-1 prefix `a.b.c.d/r` with $r > 8$ is moved to tier 2 and a new 8 bit prefix `a/8` is inserted at tier-1 (if it does not already exist).

We finally create another tree called "3-Tier" that has no more than 3 tiers. The top tier is formed by up to 256 groups with 8-bit long prefixes, tier 2 by up to 256 groups with 16-bit long prefixes, and the third tier by up to 256 groups each with a 24-bit long prefix. The mean routing table sizes for these three trees, presented in Table 4, show dramatic reduction in the number of entries that must be stored by each peer. However the stretch is significantly penalized by these transformations on the tree, as shown in Table 5. We clearly face a tradeoff between lookup latency and memory requirements.

	Mean routing table size		
Tier	Original+1	16-bit+1	3-Tier
1	143	143	143
2	436	223	105
3	831	288	13
4	1,279	428	-
5	696	556	-

Table 4. Mean routing table size in the trees where all tier-1 groups have 8-bit prefixes.

	Original+1	16-bit+1	3-Tier
stretch (TOPLUS/IP) / confidence margin	1.90 / (±0.20)	2.01 / (±0.22)	2.32 / (±0.09)

Table 5. TOPLUS vs. IP stretch in the trees where the tier-1 groups all have 8-bit prefixes..

5 Conclusion

TOPLUS takes an extreme approach for integrating topological consideration into a P2P service. TOPLUS is fully distributed, and is also symmetric in the sense that all nodes have the same role. TOPLUS bears some resemblance to Pastry [3, 6] and Tapestry [4]. In particular, Pastry and Tapestry also use delegate nodes and prefix (or suffix) matching to route messages. However, unlike Pastry, we map the groups directly to the underlying topology, resulting in an unbalanced tree without a rigid partitioning, and in a routing scheme that initially makes big physical jumps rather than small ones. We have shown that TOPLUS offers excellent stretch properties, resulting in an extremely fast lookup service. Although TOPLUS suffers from some limitations, which we have exposed and discussed, we believe that its remarkable speed of lookup and its simplicity make it a promising candidate for large-scale deployment in the Internet. Furthermore, TOPLUS can be employed in a straightforward manner to implement on-demand P2P caching of data in ISPs or corporate networks, and can serve as a benchmark for measuring the performance of other lookup services.

References

1. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. ACM SIGCOMM*, 2001.
2. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM*, 2001.
3. 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)*, (Heidelberg, Germany), pp. 329–350, November 2001.
4. B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr 2001.
5. S. Shenker, S. Ratnasamy, M. Handley, and R. Karp, "Topologically-aware overlay construction and server selection," in *Proceedings of Infocom'02*, (New York City, NY), 2002.
6. M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Topology-aware routing in structured peer-to-peer overlay networks," Tech. Rep. MSR-TR-2002-82, Microsoft Research, One Microsoft Way, Redmond, WA 98052, 2002.
7. A. D. Joseph, B. Y. Zhao, Y. Duan, L. Huang, and J. D. Kubiatowicz, "Brocade: Landmark routing on overlay networks," in *Proceedings of IPTPS'02*, (Cambridge, MA), Mar. 2002.
8. H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin, "The impact of routing policy on internet paths," in *INFOCOM*, pp. 736–742, 2001.
9. M. Freedman and D. Mazieres, "Sloppy hashing and self-organizing clusters," in *Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Mar. 2003.
10. B. Krishnamurthy, J. Wang, and Y. Xie, "Early measurements of a cluster-based architecture for P2P systems," in *ACM SIGCOMM Internet Measurement Workshop*, (San Francisco, CA), Nov. 2001.
11. B. Krisnamurthy and J. Wang, "On network-aware clustering of web sites," in *Proc. SIGCOMM 2000*, Aug. 2000.
12. J. Wang, *Network Aware Client Clustering and Applications.* PhD thesis, Cornell University, May 2001.
13. P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer informatic system based on the XOR metric," in *Proceedings of IPTPS'02*, (Cambridge, MA), Mar. 2002.
14. M. Waldvogel, *Fast Longest Prefix Matching: Algorithms, Analysis, and Applications.* Aachen, Germany: Shaker, Apr. 2000.
15. J. Kangasharju and K. W. Ross, "Adaptive replication and replacement strategies for P2P caching." unpublished, July 2002.
16. "http://rv-archive.uoregon.edu/." Oregon University "Route Views" archive.
17. "http://www.merit.edu/∼ipma/routing_table/." Merit Network, Internet Performance Measurement and Analysis (IPMA) Project.
18. "http://www.castify.net." Castify Networks.
19. "http://www.arin.net/whois/arinwhois.html." Whois service at RIPE.
20. "http://moat.nlanr.net/ipaddrocc/." NLANR.
21. "http://www.traceroute.org." Traceroute site.
22. K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts.," in *Proceedings of 2nd Internet Measurement Workshop 2002*, (Marseille, France), November 2002.
23. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen, "PlanetP: Infrastructure support for P2P information sharing," Technical Report DCS-TR-465, Department of Computer Science, Rutgers University, 2002.