Topology-Centric Look-Up Service

Keith W. Ross, Ernst W. Biersack, Pascal Felber, Luis Garces-Erice, Guillaume Urvoy-Keller

I. Introduction

Several important proposals have been put forth for providing a distributed P2P look-up service, including Chord [1], CAN [2], Pastry [3] and Tapestry [4]. These look-up services can be compared in many ways, including speed of look-up, implementation complexity, symmetry, potential for caching, and resilience to faults and attacks.

It turns out that for many measures — including speed of look-up and potential for caching — it is highly desirable that the look-up 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 look-up services that take topology into special consideration [5][6][7].

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

1) How can we design a P2P look-up 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 look-up 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 are derived directly from the network prefixes contained in BGP tables and elsewhere. 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 look-up service. At the very least, it serves as a benchmark against which other look-up services can compare their stretch and caching performance. It admittedly has some drawbacks with respect to non-uniform population of the node id space, load balancing, and correlated node failures. In the second half of this paper we will describe how TOPLUS can be modified to alleviate these problems.

A. Related Work

The paper [5] discusses how the original CAN design can be modified to account for topological considerations. The 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 high in their simulation results.

The paper [6] examines the topological properties of a modified version of Tapestry. In this design, a message typically takes small topological steps initially and big steps at the end of the route. We'll 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 look-up services, it still reports an average stretch of 2.2 when the Mercator topology [8] model is used.

Cluster-based Architecture for P2P (CAP) [9] is a P2P architecture that has built from the ground up with topo-

All the authors are with Institut Eurecom, Sophia Antipolis, France, email: {ross,erbi,felber,garces,urvoy}@eurecom.fr

logical 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 [10] [11] are currently among the most promising techniques to create the groups in TOPLUS.

II. 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 look-up 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 very close" to n_d; (2) After arriving at n_1, the message remains topologically very close to n_d as it is routed through the subsequent intermediate nodes, until it finally reaches n_d. Clearly, if the look-up 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.[1] 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).

[1] 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.

The collection of sets $\mathcal{G}$ could be created by collecting the IP prefix networks from BGP tables and/or other sources [10] [11]. 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, although promising, would likely require that sets be massaged so that the four properties of a proper nesting are satisfied. Additionally, some of the inner groups may be combined, or absorbed into larger groups, in order to reduce the number of groups and to make sure that each of the inner groups almost always contains several up nodes (although these up nodes can change over time). And yet additionally, it may be desirable to eliminate all the groups in the top one or two tiers, in order to improve look-up speed (see below). As part of our on-going research, we are using BGP prefixes to build a proper nesting that is optimized for P2P look-up.

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.

It is straightforward to see that if $\mathcal{G}$ be 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 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 in the obvious manner (see Figure 1). Let L denote the number of tiers. Note that each of the leaf groups in the partial order tree is an inner group.

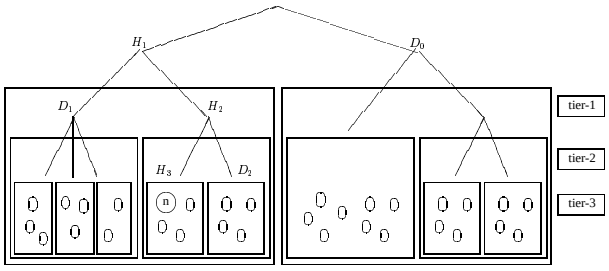

Fig. 1. A TOPLUS hierarchy.

A. Node State

Each node (that is, peer) n (with its IP address) is contained in a collection of telescoping sets $\mathcal{H}(n) = \{H_l, H_{l-1}, \cdots, H_1\}$ where $H_l \subset H_{l-1} \subset \cdots \subset H_1$ for some $l \in \{1, \ldots, L\}$. Except for H_l (which is an inner group), each set in $\mathcal{H}(n)$ has one or more direct descendants in the partial-order tree (see Figure 1). (By "direct" we mean one hop below in the partial-order tree.) Let $\mathcal{D}(n)$ be the collection of all the direct descendants generated from the sets in $\mathcal{H}(n)$, not including the sets in $\mathcal{H}(n)$.

For each set $G \in \mathcal{D}(n)$, node n should know the IP address of a node n' in G. Such a node n' is said to be *n's delegate node in G*. The list of all the IP addresses for n's delegate nodes along with IP address of node n itself is called node n's **routing table**. In the simple example of Figure 1, node n has three IP addresses in its routing table: the IP address of itself, of a node in D_2, and of a node in D_1. Each node n should also know the IP addresses of all the up nodes in its inner group. The list of all the IP addresses in a node's inner group is called the **inner-group table**.

B. Prefix-Routing Look-Up

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 is hence 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.

Each node n has a look-up API. Node X inputs into the API a key k (that is, 32-bit suffix of k'), and the API returns the up node Y that, among all the up nodes, has the closest IP address to the key.

Suppose node X wants to look-up k. Node X examines its routing table and performs a longest-prefix match, comparing k with all the entries in X's routing table. Suppose node X' has the longest prefix. Then node X forwards the message to X'. The process continues, using longest-prefix matching to forward the messages from peer to peer. *Thus, in TOPLUS, every peer mimics an IP router*.

The process continues until the message reaches a node Z such that the longest prefix match in Z's routing table is with the IP address of Z itself. When this occurs, Z routes the message to the node in its inner-group table that has the closest IP address to k.

When a new node X joins the system, X asks an arbitrary exiting node, say Y, to determine (using TOPLUS) the closest node to X (using X's IP address as the key). Denote this closest node by Z. Node X then initializes its routing and inner-group tables with Z's routing and inner-group tables. Node X's routing table should then be modified to satisfy a "diversity" property, which we'll discuss below. Also, a small fraction of existing nodes tables should be modified when X joins.

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.

That completes our description of TOPLUS. TOPLUS is fully distributed, and is also symmetric in the sense that no node is a supernode. Because the groups and their nestings have been derived directly from the underlying topology, it 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 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, particularly if top tiers of the partial-order tree are eliminated from the nesting. Once the message arrives at G, it will remain in G until it arrives at n_d.

As mentioned above, each node in TOPLUS mimics a router in the sense that it routes messages based on longest-prefix matching of IP addresses. To this end, nodes can use highly-optimized longest-prefix matching schemes [12] deployed in high-speed routers for routing messages. Now consider how many IP addresses are in a node's routing table. Assuming that the node's inner group is in the lowest tier, L, let $d_{L-1}, d_{L-2}, \ldots, d_1$, be the number of direct descendants derived from the partial order tree. Then the total number of IP addresses in the node's routing table is $d = 1 + d_1 + \cdots + d_{L-1}$. During the formation of the nested groups, it is desirable to prevent d 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.

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 Tapestry, 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.

C. 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 X wants to obtain the file f associated with key k. Once X learns from the look-up service

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

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 look-up service. This is done as follows. Let the network prefix for G be denoted by w.x.y.z/r. Now suppose some node $X \in G$ wants to find the file f associated with key $k \in I$. Then X 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 X then inserts a message with key k_G into TOPLUS. The look-up service will return to X the node Y_G that is responsible for k_G. Node Y_G will be in G, and all the messages traveling from X to Y_G will be confined to G. X then asks Y_G for f. If Y_G has f (cache hit), then Y_G will send f to X at a relatively high rate. If Y_G does not have f (cache miss), Y_G will use TOPLUS to obtain f from the global look-up service. After obtaining f, Y_G will cache f in its local shared storage and pass a copy of f to X. (X does not make f available through caching since it is not responsible for k within G or globally.)

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 [13] 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.

D. Node Joins and Departures

When a node joins or leaves, tables need to be updated for a small fraction of the other existing nodes. These update operations can be done aggressively or lazily, with many different variations [1], [2], [4], [3]. The particular update scheme is not central to TOPLUS.

However, for robustness to node failures, a "diversity" property should be maintained across the tables in different nodes. To describe this diversity property, consider two arbitrary nodes m and n in the same inner group. For these two nodes, we have $\mathcal{D}(m) = \mathcal{D}(n)$, and thus both nodes will have the same number of entries in their routing tables. For a given group $G \in \mathcal{D}(m)$, node m will have in its routing table the IP address of a delegate node $m' \in G$, and node n will the IP address of a delegate node $n' \in G$. The diversity property is that m and m' are different with high probability. This diversity property can be maintained by providing nodes replacement delegates, either at node arrivals or during the look up process itself.

III. 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 P2P look-up service. We now discuss some of the drawbacks of the TOPLUS design. We'll modify the TOPLUS design 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 will 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 look-up services CAN, Chord, Pastry and Tapestry 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 look-up 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 are lost. 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 address, we assign each node a permanent "virtual id," where the "virtual id" is uniformly distributed over the IPv4 address space. 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 look-up process continues until the message reaches a node Z such that the longest prefix match is with the IP address of Z itself. But once the message is at Z, node Z now determines, among all the virtual ids in its inner group table, the virtual id that is the closest to the key. Z then sends the message to the node corresponding to this virtual id. Note that this enhancement also provides load balancing across the inner group.

Another variation is for each inner group to use its own look-up service locally within the inner group. For example, Chord, CAN, Pastry, or Tapestry could be used in each inner group. For sake of discussion, suppose that Chord is used in each inner group. When the message with key k arrives at Z in inner group G, TOPLUS passes k to the "local Chord look-up service" to determine the node in G responsible for k. All the functionality and features of Chord could be used in this local group.

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. And 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. For a given key k, it is possible to define for k a first-place inner group, a second-place inner group, etc, all of which can be determined directly from k using TOPLUS. The key data pair would then be replicated in multiple nodes in each of the, say, top K groups. These top-K groups can track each other, so that if one fails, a new top group can be identified and the key/data pairs can be copied into the new group. There will not be separate clique of K groups for each key, since each clique will typically cover a large number of keys.

IV. Final Remarks

TOPLUS takes an extreme approach for integrating topological consideration into a P2P service. It serves as a benchmark for measuring the performance of other look-up services, and it raises many issues for debate. In the spirit of the workshop, we have introduced TOPLUS as a work in progress. We are currently building proper nestings and testing the performance of TOPLUS for the nestings. (Many of these results will be available before the date of the workshop.) We are also comparing in more detail TOPLUS to the recent proposals for adding topological considerations to Tapestry and CAN [5][6].

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," *Lecture Notes in Computer Science*, 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] 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.

[10] B. Krisnamurthy and J. Wang, "On network-aware clustering of web sites," in *Proc. SIGCOMM 2000*, Aug. 2000.

[11] J. Wang, *Network Aware Client Clustering and Applications*. PhD thesis, Cornell University, May 2001.

[12] M. Waldvogel, *Fast Longest Prefix Matching: Algorithms, Analysis, and Applications*. Aachen, Germany: Shaker, Apr. 2000.

[13] J. Kangasharju and K. W. Ross, "Adaptive replication and replacement strategies for P2P caching." unpublished, July 2002.