SkipNet: A Scalable Overlay Network with Practical Locality Properties

Nicholas J.A. Harvey[*][†], Michael B. Jones[*], Stefan Saroiu[†], Marvin Theimer[*], Alec Wolman[*]

Abstract: *Scalable overlay networks such as Chord, Pastry, and Tapestry have recently emerged as a flexible infrastructure for building large peer-to-peer systems. In practice, two disadvantages of such systems are that it is difficult to control where data is stored and difficult to guarantee that routing paths remain within an administrative domain. SkipNet is a scalable overlay network that provides controlled data placement and routing locality guarantees by organizing data primarily by lexicographic key ordering. SkipNet also allows for both fine-grained and coarse-grained control over data placement, where content can be placed either on a pre-determined node or distributed uniformly across the nodes of a hierarchical naming subtree. An additional useful consequence of SkipNet's locality properties is that partition failures, in which an entire organization disconnects from the rest of the system, result in two disjoint, but well-connected overlay networks.*

1 Introduction

Scalable overlay networks, such as Chord [21], CAN [16], Pastry [18], and Tapestry [26], have recently emerged as flexible infrastructure for building large peer-to-peer systems. A key function that these networks enable is a distributed hash table (DHT), which allows data to be uniformly diffused over all the participants in a peer-to-peer system.

While DHTs provide nice load balancing properties, they do so at the price of controlling where data is stored. This has at least two disadvantages: data may be stored far from its users and it may be stored outside the administrative domain that it belongs to. This paper introduces SkipNet, a scalable overlay network that supports both traditional overlay functionality, including scalability and load balancing, as well as two locality properties that we refer to as *content locality* and *path locality*.

Content locality refers to the ability to either explicitly place data on specific overlay nodes or distribute it across nodes within a given organization. Path locality refers to the ability to guarantee that message traffic between two overlay nodes within the same organization is routed within that organization only.

Content and path locality provide a number of advantages for data retrieval, including improved availability, performance, manageability, and security. For example, an organization can still access and update important data even when disconnected from the rest of the Internet. Similarly, data needed for autonomous activities within a campus or a building can be stored locally to preserve autonomy. Because data can be stored near clients that use it, performance and manageability are also enhanced. Explicit *content placement* onto a specific overlay node also enables provisioning of that node to reflect anticipated popularity. Similarly, content placement ensures administrative control over issues such as scheduling maintenance for machines storing important data.

Content locality also allows one to control which administrative domain data resides in. Even when encrypted and digitally signed, data stored on an arbitrary overlay node in a foreign administrative domain is susceptible to denial of service (DoS) attacks as well as traffic analysis. Although techniques for making current DHT schemes resilient against DoS attacks exist [4], it is not clear whether participating organizations would feel comfortable about storing important data on machines outside of their direct control.

Controlling content placement is in direct tension with the goal of a DHT, which is to uniformly distribute data across a system in an automated fashion. A generalization that combines these two notions is *constrained load balancing*, in which data is uniformly distributed across a well-defined subset of the nodes in a system, such as all nodes in a single organization, all nodes residing within a given building, or all nodes residing within one or more data centers.

Even if the desired content locality could be achieved, existing overlay networks still forward traffic through intermediary nodes that could be in arbitrary administrative domains. Although some overlay designs [5] are likely to keep routing messages localized most of the time, none can *guarantee* path locality—that messages between two nodes within the same organization are routed within that organization only. For example, without such a guarantee

[*]Microsoft Research, Microsoft Corporation, Redmond, WA, 98052, {nickhar, mbj, theimer, wolman}@microsoft.com

[†]Department of Computer Science and Engineering, University of Washington, Seattle, WA, {nickhar, tzoompy}@cs.washington.edu

the route from *explorer.ford.com* to *mustang.ford.com* could pass through *camaro.gm.com*, a scenario that people at *ford.com* might prefer to prevent. The provision of path locality is thus important for both availability and security reasons.

SkipNet employs two address spaces for routing: a lexicographic space as well as a numeric space. Node names and content identifier strings are mapped directly into the lexicographic space, while hashes of the node names and content identifiers are mapped into the numeric space. The lexicographic space is a distributed generalization of Skip Lists [15] and it is used to support content placement, path locality, and efficient range queries. The numeric space is used to support DHT functionality and efficient message routing between overlay nodes. A combination of the two spaces is used to support constrained load balancing.

A useful bonus of SkipNet's locality properties is resiliency against common Internet failures. Because SkipNet clusters nodes according to their lexicographic name ordering, names within a single organization survive failures that disconnect the organization from the rest of the Internet. In the case of independent failures, SkipNet has similar resiliency to previous overlay networks [21].

The basic SkipNet design, not including its enhancements to support constrained load balancing, network proximity-aware routing, and reduced overhead for virtual nodes, has been concurrently and independently invented by Aspnes and Shah [1]. As described in Section 2, their work has a substantially different focus than our work and the two efforts can be viewed as being complementary to each other while still starting from the same underlying inspiration.

The rest of this paper is organized as follows: Section 2 describes related work. Section 3 describes SkipNet's basic design, Section 4 discusses SkipNet's properties, and Section 5 presents enhancements over the basic design. Section 6 discusses design alternatives to SkipNet, Section 7 presents an experimental evaluation, and Section 8 concludes the paper.

2 Related Work

A large number of peer-to-peer overlay network designs have been proposed recently, each making different trade-offs with respect to system scalability, security, anonymity, retrieval accuracy, fault tolerance, and content availability. Our focus is on building a general-purpose, scalable, fault-tolerant overlay that allows for explicit control over content availability and placement. In consequence, SkipNet should be viewed as an alternative to general-purpose overlays such as CAN [16], Chord [21], Pastry [18], and Tapestry [26].

The key feature of systems such as CAN, Chord, Pastry, and Tapestry is that they afford scalable routing paths while maintaining a scalable amount of routing state at each node. By scalable routing path we mean that the expected number of forwarding hops between any two communicating nodes is small with respect to the total number of nodes in the system. Chord, Pastry, and Tapestry scale with $\log N$, where N is the system size, while maintaining $\log N$ routing state at each overlay node. CAN scales with $N^{1/D}$, where D is a dimensionality factor with a typical value of 6, while maintaining an amount of per-node routing state proportional to D.

A second key feature of these systems is that they are able to route to destination addresses that do not equal the address of any existing node. Each message is routed to the node whose address is "closest" to that specified in the destination field of a message. This feature enables them to implement a distributed hash table (DHT) [14], in which content is stored at an overlay node whose node ID is closest to the result of applying a collision-resistant hash function to that content's name, or another property of it. Distributed hash tables have been used, for instance, in constructing the Past [19] and CFS [7] distributed filesystems, the Overlook [23] scalable name service, the Squirrel [10] cooperative web cache, and scalable application-level multicast [6, 20, 17].

Our work borrows ideas from the Skip List [15] data structure, applying them to construct a new kind of scalable overlay network called SkipNet. We also borrow concepts from both Chord and Pastry. The ideas derived from skip lists enable SkipNet to support efficient range queries, content placement, and path locality. The ideas taken from Chord and Pastry enable DHT functionality and efficient message routing between overlay nodes. A combination of these ideas is used to support constrained load balancing.

The fundamental philosophy of systems such as Chord and Pastry is to diffuse content randomly throughout an overlay in order to obtain uniform, load-balanced, peer-to-peer behavior. The fundamental philosophy of SkipNet is to enable systems to preserve useful content and path locality, while still enabling load balancing over constrained subsets of participating nodes.

This paper is not the first to observe that locality properties are important in peer-to-peer systems. Keleher et al. [12] make two main points: DHTs destroy locality and they discard useful application-specific information. Vahdat et al. [24] raises the locality issue as well. Both of these problems are addressed by SkipNet. By using names rather than hashed identifiers to order nodes in the overlay, natural locality based on the names of objects is preserved. Furthermore, by arranging content in name order rather than dispersing it, operations on ranges of names are possible in SkipNet.

Aspnes and Shah have independently invented the same basic data structure that defines a SkipNet [1]. The focus of their work is primarily on supporting queries based on key ordering and on describing precisely how their data structure can be maintained in the face of concurrent random node joins, departures, and failures. In contrast, our work is focused primarily on the content and path locality properties of the design. SkipNet also includes several extensions beyond the basic design that it shares with Aspnes and Shah's work: Constrained load balancing is supported through a combination of searches in both the lexicographic and numeric address spaces. Network proximity-aware routing is obtained by means of two auxiliary routing tables. Finally, SkipNet is also able to support hosting virtual nodes on a single physical node with substantially less overhead than other existing overlays.

3 Basic Structure

In this section, we introduce the basic design of SkipNet. We present the SkipNet architecture, including how to route in SkipNet, and how to join and leave a SkipNet.

3.1 Skip Lists

A skip list, first described in Pugh [15], is a dictionary data structure typically stored in-memory. A skip list is a sorted linked list in which some nodes are supplemented with pointers that skip over many list elements. A "perfect" skip list is one where the height of the i'th node is the exponent of the largest power-of-two that divides i. Figure 1 depicts a perfect skip list. Note that pointers at level h have length 2^h (i.e. they traverse 2^h nodes in the list). A perfect skip list supports searches in $O(\log N)$ time.

Because it is prohibitively expensive to perform insertions and deletions in a perfect skip list, Pugh suggests a probabilistic scheme for determining node heights while maintaining $O(\log N)$ searches with high probability. Briefly, each node chooses a height so that the probability of choosing height h is $1/2^h$. Thus, with probability $1/2$ a node has height 1, with probability $1/4$ it has height 2, and so forth. Figure 2 depicts a probabilistic skip list.

3.2 SkipNet Data Structures

Whereas skip lists are an in-memory data structure that is traversed from its beginning, we want a

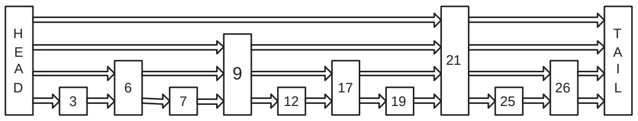

Figure 1. *A perfect skip list.*

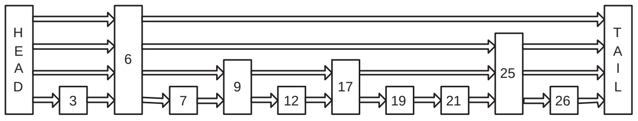

Figure 2. *A probabilistic skip list.*

data structure that links together distributed computer nodes and supports traversals that may start out from any node in the system. Furthermore, because peers should have uniform roles and responsibilities in a peer-to-peer system, we want the state and processing overhead of all nodes to be roughly the same. In contrast, skip lists maintain a variable number of pointers per data record and consequently experience a substantially different amount of traversal traffic at each data record.

3.2.1 Lexicographic Address Space

The key idea we take from skip lists is the notion of maintaining a sorted list of all data records as well as pointers that "skip" over varying numbers of records. We transform the concept of a skip list to a distributed system setting by replacing data records with computer nodes, using the string names of the nodes as the data record keys, and forming a ring instead of a list. The ring must be doubly-linked to enable path locality, as is explained in Section 3.3.

Rather than having nodes store a variable number of skip pointers, each node stores $2 \cdot \log N$ pointers, where N is the number of nodes in the overlay system. The pointers at level i of a given node's routing table point to nodes that are 2^i nodes to the left and right of the given node. Figure 3 depicts a SkipNet containing 8 nodes and shows the routing table pointers that nodes A and O maintain.

The SkipNet in Figure 3 is a "perfect" SkipNet: its pointers traverse exactly 2^h data records in all cases and it suffers from the same insertion and deletion overheads as perfect skip lists. We derive a probabilistic SkipNet design by first observing that the pointers maintained at each level of a node's routing table can be thought of as forming multiple rings of nodes, as depicted in Figure 4.

Figure 4 shows the routing table pointers of the overlay network of Figure 3, arranged to show node interconnections at every level of a node's routing table. All nodes are connected by the ring that is formed from the level 0 routing table pointer on each node. Level 1 table entries point to nodes that are 2 nodes

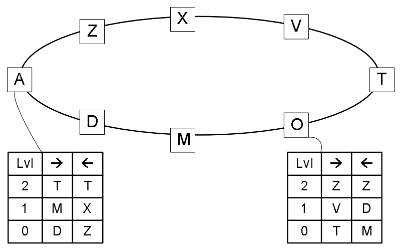

Figure 3. *Lexicographically ordered SkipNet nodes. Routing tables of nodes A and O are shown.*

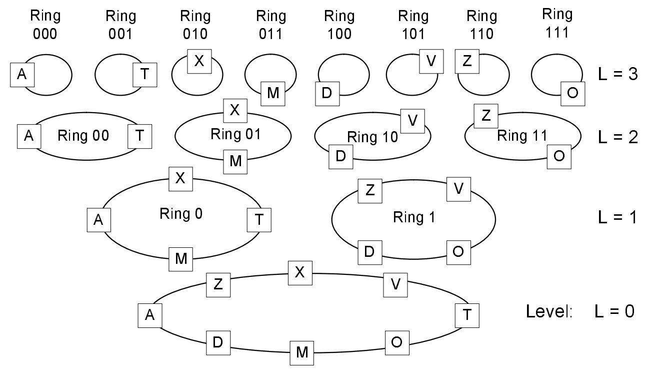

Figure 4. *The full SkipNet routing infrastructure for an 8 node system, including the ring labels.*

away and hence divide the overlay nodes into two disjoint sets. Similarly, routing table entries at level 2 form 4 disjoint rings of nodes and so forth. Note that rings at level $h + 1$ are obtained by splitting a ring at level h into two disjoint sets, each ring containing every second member of the level h ring.

Routing is possible in $O(\log N)$ forwarding hops because the rings at level h skip over 2^h nodes. We can achieve the same routing performance with high probability as long as ring memberships include roughly every 2^h-th node in each ring at level h.

Consider a node that wishes to join the SkipNet. It must join the lowest level ring in any case. However, it does not matter which level 1 ring the node joins, as long as, on average, half of all nodes join each level 1 ring. Thus, the node randomly decides which level 1 ring to join, and this procedure is then followed recursively for higher level rings. In this manner, we can probabilistically ensure the desired "skip" characteristics of each ring.

3.2.2 Numeric Address Space

Figure 4 illustrates an important aspect of SkipNet: Ring membership of a node can be encoded by means of a unique binary number. The first h bits of the number determines ring membership at level h. For example, node X's memberships are represented as 010 and its membership among rings at level 2 is determined by taking the first 2 bits of 010, which designate ring 01. A node can decide which rings to join by randomly generating a binary number, which we refer to as the node's *random ID*. As described in [21], there are advantages to using a collision-resistant hash (such as MD-5) of the node's DNS name as the random ID.

The uniqueness of these numbers allows us to define a second address space for SkipNet nodes. This address space is identical to that used by traditional overlays, such as Chord and Pastry, and will enable us to provide the same DHT functionality.

Readers familiar with Chord may have observed that SkipNet's routing tables seem very similar to those maintained by Chord. There is, however, a fundamental difference. SkipNet's routing tables point into a lexicographic space populated by nodes' string names whereas Chord's routing tables point into a numeric space populated by unique hashes derived from nodes' string names. Chord obtains its $O(\log N)$ routing and node insertion properties by relying on hashing string names uniformly throughout its address space. In SkipNet, node string names are *not* spread uniformly throughout its lexicographic address space. Thus, to maintain efficiency, SkipNet defines an secondary address space in which nodes are uniformly dispersed.

3.2.3 Duplicate Pointer Elimination

A node keeps a pointer to its immediate predecessor and successor on every ring. As a result, two nodes may be neighbors on several rings and, therefore, maintain duplicate pointers. However, a duplicate pointer does not add value to a routing table. Replacing it with a suitable alternative, such as the following neighbor in the relevant ring, can improve routing performance by a moderate amount (typically around 20%). Routing table entries that have been adjusted in this fashion can only be used to route lexicographic search queries since they violate the requirement that a node point to its closest neighbor on a ring.

3.3 Lexicographic Searching

Lexicographic searches in SkipNet are based on the same basic principle as searches in skip lists: Follow pointers that route closest to the intended destination. At each node, a search message will be routed along the highest-level pointer that does not point past the destination value. Routing terminates when the message arrives at a node whose lexicographic name is closest to the destination. Figure 5 illustrates this algorithm.

Since nodes are lexicographically ordered along each ring and a message is never forwarded past its destination, the pointers followed are all lexicographically placed between the source and the destination. Thus, when a lookup message originates at a node

```
// Called only at the node that originates the search
Node Search(string destID) {
   if (destID < currNode.LexID) then
      node = SearchClockwise(destID)
   else if (destID > currNode.LexID)
      node = SearchCounterClockwise(destID)
   return node;
}

// Called at all intermediate hops along the search path
Node SearchClockwise(string destID) {
   h = currNode.maxHeight
   while (h > 0)
      nextNode = currNode.ClockwiseFinger[h]
      // Check if the nextNode.LexID lies in between currNode
      // and the destination, when traversing the ring clockwise
      if (LiesBetweenClockwise(nextNode.LexID,
                               currNode.LexID, destID)) then
         return SendMessage(MsgSearchClockwise, nextNode, destID)
      h = h - 1
   endwhile
   // At h=0 the destination is between currNode and nextNode,
   // so we return the closest to the destination
   return ClosestNode(destID, currNode.LexID, nextNode.LexID)
}
```

Figure 5. *The SkipNet lexicographic search algorithm (clockwise).*

whose name shares a common prefix with the destination, all nodes traversed by the message have names that share the same prefix as the source and destination do. Note that, because rings are doubly-linked, this scheme can route using both right and left pointers depending upon whether the source name is lexicographically smaller or greater than the destination, respectively. The key observation of this scheme is that a lexicographic search is routed along nodes with non-decreasing prefix matches with the destination.

If the source name and the destination share no common prefix, a search message could be routed in either direction, using right or left pointers. For fairness sake, one could randomly pick a direction to go so that nodes whose names are in the middle of the alphabet do not get a disproportionately larger share of the forwarding traffic than do nodes whose names are near the beginning and end of the alphabet. For simplicity however, our current implementation never wraps around from Z to A or vice-versa.

The expected number of hops traversed by a search message is $O(\log N)$ with high probability. For a rigorous and complete proof see Harvey et al. [9].

3.4 Node Join and Departure

As outlined in Section 3.2.2, in order to join a SkipNet, a newcomer must first find the top-level ring that corresponds to the random ID obtained by hashing the newly-joining node's name with a collision resistant hash function. The routing algorithm that finds the top-level ring for a newcomer applies two simple steps repeatedly. For each node encountered in SkipNet, it first determines the highest level ring shared between the newcomer and the node. It then traverses this ring until it finds a new node that shares a higher level ring with the newcomer. This procedure is repeated until the newcomer finds no other node with which it

```
//  - The first phase locates the highest-level ring that
//    contains the newNode and at least one other node.
//  - The second phase works its way from that top-level ring,
//    inserting itself on one ring at each height.

Init() {
   phase = upward
   currentH = 0
}

Insert() {
   if (phase == upward) then
      nextHop = NextHopUp()
   else
      nextHop = NextHopDown()
   if (phase != complete) then
      SendMessage(InsertMsg, nextHop)
}

Node NextHopUp() {
   h = LongestCommonPrefix(currNode.RandID, newNode.RandID);
   if (h > currentH) then
      currentH = h
      ringStart = currNode

   nextHop = currNode.ClockwiseFinger[h]
   if (ringStart == nextHop) then
      phase = downward
      return NextHopDown()
   else
      return nextHop
}

Node NextHopDown() {
   while (currentH > 0)
      nextHop = currNode.ClockwiseFinger[currentH]
      if (LiesBetweenClockwise(newNode.LexID, currNode.LexID,
         nextHop.LexID)) then
         InsertHere(newNode)
      else
         return nextHop
      currentH = currentH - 1
   endwhile
   phase = complete
   return null
}
```

Figure 6. *SkipNet node insertion algorithm.*

shares a higher level ring. As described in Harvey et al. [9], finding the top-level ring takes $O(\log N)$ message steps to complete, with high probability.

This top-level ring is the first one that the newcomer joins, using a lexicographic sequential search within this ring only. Once the join is complete, the new node uses its neighbors to search in the next lower level ring for its lexicographic position and join it accordingly. This process is repeated for each level until the newcomer joins the level-0 ring. For correctness, none of the existing nodes point to the newcomer until the new node joins the level-0 ring. Only then does the newcomer send messages to its neighbors along each ring to indicate its presence.

The key observation for this algorithm's efficiency is that a newcomer joins a ring at a certain level only after joining a higher level ring. As a result, the sequential lexicographic search within the ring to be joined will typically not traverse all members of the ring. Instead, the range of nodes traversed is limited to the range between the newcomer's neighbors at the higher level. Therefore, with high probability, a node join in SkipNet will traverse $O(\log N)$ hops (for a proof see Harvey et al. [9]). Figure 6 shows the insertion algorithm; *upward* refers to the phase in which a newcomer finds the top-level ring to join; *downward* refers to the phase in which it joins SkipNet rings at decreasing levels successively.

The basic observation in handling node departures is that SkipNet can route correctly as long as the bottom level ring is maintained. All pointers but the level-0 ones can be regarded as routing optimization hints, and thus not necessary to maintain routing protocol correctness. Therefore, like Chord and Pastry, SkipNet maintains and repairs these rings' memberships lazily, by means of a background repair process.

To maintain the bottom ring correctly, each SkipNet node maintains a *leaf set* that points to additional nodes along the bottom ring. In our current implementation we use leaf set size of 16, just as Pastry does.

3.5 Numeric Searching

SkipNet can implement DHT-like functionality by routing messages in its numeric address space. Routing in the numeric address space is equivalent to the *upward* phase of node joins, in which a newcomer searches for a ring with a particular random ID. Consequently, the message routing cost for numeric searches is $O(\log N)$ [9].

Some intuition for why SkipNet can support both lexicographic and numeric routing with the same data structure is illustrated in Figure 4. Since SkipNet is based on skip lists, the nodes in the bottom ring are sorted by the lexicographic ordering. Simultaneously, the SkipNet ring structure also forms a trie [8] of the random IDs, and thus nodes are ordered by their random IDs across the top rings. The rings at intermediate levels are partially sorted by both lexicographic and random IDs.

4 Useful Locality Properties of SkipNet

The previous section described the data structure and protocol that implement SkipNet. In this section we discuss the useful locality properties that SkipNet is able to provide.

4.1 Content and Routing Path Locality

Because SkipNet employs lexicographic searching, it allows for control over data placement. Incorporating a peer's address into a content name guarantees that the content will be hosted on that respective peer. Although it could use arbitrary naming conventions for node addresses and content names, SkipNet's current implementation uses DNS names. For user convenience, it also reverses the order of DNS names, so that *john.microsoft.com* will be interpreted as *com.microsoft.john*. As an example, when a document, *doc-name*, should be stored on *john.microsoft.com*, naming it *john.microsoft.com/doc-name* is sufficient to guarantee that it will be stored onto the desired machine.

SkipNet's lexicographic ordering of nodes also enables it to guarantee routing locality, as described in Section 3.3. For example, a message originating from *john.microsoft.com* for *jane.microsoft.com* will never leave *microsoft.com* (assuming that only Microsoft nodes may use that naming prefix).

4.2 Constrained Load Balancing

Because SkipNet intertwines both a lexicographic space and a numeric space, it is able to provide constrained load balancing. That is, it is able to support the functionality of multiple DHTs of differing scopes within a single overlay network. This is substantially cheaper than maintaining a separate overlay network for each DHT. It is also much simpler in that overlay nodes do not need to know which DHTs they need to belong to; the scope and node membership of a DHT is determined strictly by the choice of data object names that clients choose.

Constrained Load Balancing (CLB) involves dividing a data object's name into two parts: a part that specifies the set of nodes over which DHT load balancing should be performed and a part that is used as input to the DHT's hash function. In SkipNet the special character '*!*' is used as a delimiter between the two parts of the name. For example, the name *msn.com/DataCenter!TopStories.html* indicates load balancing over nodes whose names begin with the prefix *msn.com/DataCenter*. The suffix, *TopStories.html*, used as input to the load balancer hash function, determines which node the data object actually ends up on.

CLB is implemented by searching for the name prefix in lexicographic space and then switching to the numeric space to search for the hash of the name suffix. The search for the hash of the suffix is constrained to only consider nodes that have the same name prefix as the data object's name.

As described in Section 3, searches in both the lexicographic and numeric spaces take $O(\log N)$ message hops to complete. Placing a lexicographic name prefix constraint on the numeric address space search does not change the performance characteristics. This is because numeric searches involve searching "upward" through the routing rings and the probability of finding a node on a ring that will enable upward progress is independent of the lexicographic name prefix constraint; that is, it is an independent, random variable. Because the rings are sorted, all ring members that obey the constraint are contiguous and hence can be efficiently traversed until a suitable node is found.

Note that both traditional system-wide DHT semantics as well as explicit content placement are special cases of constrained load balancing: system-wide DHT semantics are obtained by placing the hashing delimiter at the beginning of a document name. Omission of the hashing delimiter plus choosing the name of a data object to have a prefix that matches the name of a particular overlay node will result in the object being placed on that overlay node.

Constrained load balancing has its limitations. It can be performed over any naming subtree of the SkipNet but not over an arbitrary subset of the nodes of the overlay network. In this respect it has flexibility similar to a hierarchical file system's. Another limitation is that the domain of load balancing is encoded in the name of a data object. Thus, transparent remapping to a different load balancing domain is not possible.

4.3 Fault Tolerance

Previous studies [13, 2] have shown that network connectivity failures in the Internet today are due primarily to Border Gateway Protocol (BGP) misconfigurations and faults, as well as hardware, software and human failures. As a result, node failures in overlay systems are not independent, but instead, nodes belonging to the same organization or AS domain tend to fail together. In consequence, we have focused the design of SkipNet's fault-tolerance to handle two different scenarios: (1) independent failures and (2) failures occurring along organizational boundaries.

4.3.1 Independent Failures

As with Chord, the key observation in failure recovery is that maintaining correct neighbor pointers in the level 0 ring is enough to ensure correct functioning of the overlay. Since each node maintains a leaf set of l level 0 neighbors, level 0 ring pointers can be repaired by replacing them with the leaf set entries that point to the nearest live nodes following the failed node.

Like Chord, SkipNet employs a lazy stabilization mechanism that gradually updates all necessary routing table entries in the background to reflect the disappearance of n. Note that during this stabilization period, no ongoing queries are disrupted, although they might be sub-optimally routed.

4.3.2 Failures along Organization Boundaries

In previous peer-to-peer overlay designs [16, 21, 18, 26], node placement in the the overlay topology is determined by a randomly chosen numeric ID. As a result, nodes near each other in terms of network proximity are placed uniformly throughout the address space of the overlay. While a uniform distribution enables the $O(\log N)$ routing performance of the overlay it makes it difficult to predict the effect of physical link failures on the overlay network. In particular, the failure of a network link will manifest itself as multiple, randomly scattered link failures in the overlay. Taken to an extreme, it is possible for each node within a single organization that has lost connectivity to the Internet to become disconnected from the entire overlay and from all other nodes within the organization.

Since SkipNet preserves node locality within the overlay, local failures do not fragment the overlay, but instead result in ring segment partitions. Consequently in SkipNet a significant fraction of routing table entries of the disconnected nodes still point to live nodes within the same network partition. This property allows SkipNet to gracefully survive failures along organization boundaries.

4.4 Range Queries

Since SkipNet's design is based on and inspired by skip lists, it inherits their functionality and flexibility in supporting efficient range queries. In particular, since keys are stored in lexicographic order, documents sharing common prefixes are stored over contiguous ring segments. Answering range queries in SkipNet is therefore equivalent to routing along the corresponding ring segment. Because our current focus is on SkipNet's architecture and locality properties, we do not discuss range queries further in this paper.

5 SkipNet Enhancements

This section presents several optimizations to the basic SkipNet design.

5.1 Efficient Routing in Lexicographic Space

In SkipNet, the neighbors of a node are determined by the random choice of ring memberships and the ordering of identifiers in those rings. Accordingly, the overlay is constructed without considering physical network topology, potentially hurting routing performance and latency.

To deal with this problem, we introduce a second routing table called the *proximity table*, or P-table for short. P-tables are inspired by Pastry's proximity-aware routing tables [5]. As described in Section 3, a SkipNet node's basic routing table entries are expected to point to nodes that are exponentially increasing distances away. Each P-table routing entry points to a node chosen from the ring segment whose endpoints are defined by two consecutive basic routing table entries. This choice gives SkipNet the flexibility of selecting routing entries according to physical network proximity.

Like Pastry, SkipNet constantly tries to improve the quality of its P-table entries, as well as adjust to node joins and departures, by means of a background stabilization algorithm. The details of SkipNet P-table construction and maintenance, along with its complexity characteristics and routing guarantees, are contained in Harvey et al. [9].

5.2 Efficient Routing in Numeric Space and Constrained Load Balancing

Since SkipNet supports search and routing via two different spaces and uses both to implement CLB, we add a third table that optimizes searches in the numeric space, much as the P-table optimizes searches in the lexicographic space. Since SkipNet's numeric space is essentially equivalent to the random ID space of Pastry, this third table has similar functionality to the routing table that Pastry maintains. CLB searches use entries in this proximity table except when they violate the CLB search constraint. In that case, CLB reverts to using the basic routing table.

5.3 Virtual Nodes

In Web-hosting scenarios, a single machine often has multiple names. In SkipNet this can be supported by allowing a physical node to run multiple SkipNet virtual nodes.

Without any modifications to the basic SkipNet structure, each virtual node has a routing table of $O(\log N)$ entries and thus $O(k \cdot \log N)$ entries is required to host k virtual nodes. The periodic routing table maintenance traffic becomes a scalability bottleneck as k becomes large.

The key observation is that SkipNet can reduce this overhead without imposing disproportionate amounts of work on any *physical* node. In particular, the routing table state for virtual nodes can be distributed in a manner similar to probabilistic skip lists. Our approach scales such that k virtual nodes require $O(k + \log N)$ state, with only a constant factor increase in search performance. For more details see Harvey et al. [9].

6 Design Alternatives

The locality properties provided by SkipNet can also be obtained by means of suitable extensions to existing overlay networks. However, we argue that SkipNet does so in a more natural and often in a more efficient manner. In this section we describe design alternatives to SkipNet and compare them with SkipNet's approach.

DHT-based overlay networks depend on random assignment of node IDs in order to obtain a uniform distribution of nodes within the address space they use. To support explicit content placement for a single data object, one may choose a node ID that directly corresponds to the hash ID of that data object. In order to assign more than one data object to the same node one could do either of the following:

- One could virtualize the overlay nodes so that each node joins the overlay once per data object. This has the disadvantage that one must maintain a separate routing table for each data object to be assigned to a given physical overlay node.
- One could employ a two-part naming scheme, as in SkipNet, wherein data object names consist of a virtual node name followed by a node-relative name. The virtual node name is hashed to obtain an ID used to select a physical node on which to place the virtual node's data. The physical node hosting this data assigns itself this hashed ID and joins the overlay. As a result, the DHT will route requests for the virtual node's data objects to it.

The cost of the first approach is prohibitive if a single node needs to store more than a few hundred data objects. The second approach is essentially equivalent to the SkipNet approach for content placement, except that both node and data object names are translated to a numeric address space instead of being used directly.

Constrained load balancing can be achieved by maintaining multiple overlay networks, one per DHT provided. This imposes a cost in terms of both routing table state and complexity, since physical nodes must maintain separate routing tables for each DHT they join. Furthermore, when a new DHT is created to load balance data across a set of nodes, all nodes must be informed about their membership in the new overlay. Worse yet, these new members will now have to inherit the burden associated with maintaining routing tables in the new DHT. In contrast, SkipNet is able to provide arbitrarily many DHTs using a single set of routing tables per node and new DHTs can be created simply by having clients create appropriately structured data object names.

Path locality could be added to a DHT-based overlay by ensuring that specially-marked local messages are not forwarded outside of a given organizational boundary. The information would be used to exclude routing table entries from use that violate the routing constraint. Unfortunately this approach cannot *guarantee* path locality: There may not be a path to a destination with the constraint applied. Furthermore, even if a path is available, it may end up being inefficient.

Overlay networks such as Pastry can partially mitigate this problem using their support for network proximity [5]. However, Pastry's network proximity support depends on having a "nearby" node to use as a "seed" node when joining an overlay. If the "nearby" node is not within the same organization as the joining node, Pastry might not be able to provide very good path locality. Furthermore, this problem is exacerbated for organizations that consist of multiple separate "islands" of nodes that are far apart in terms of network distance. In contrast, SkipNet is able to guarantee path locality, even across organizations that consist of separate clusters of nodes.

Given the naming approach suggested in Section 4.1, a simple alternative to lexicographic searches (but *not* CLB searches) through a SkipNet overlay would be to route directly with IP after a DNS lookup. SkipNet has three key advantages over this approach. First, in the presence of node failures, overlay routing provides seamless reassignment of traffic to nearby nodes within the same organization. Second, higher level abstractions such as multicast [6, 20, 17] and load-aware replication [23] are easy to implement. Third, DNS failures do not impact data availability, since no explicit name lookups are made.

7 Experimental Evaluation

To understand and evaluate SkipNet's design and performance, we used a simple packet-level, discrete event simulator that counts the number of packets sent over a physical link and assigns a constant delay to each link [9]. It does not model either queuing delay or packet losses because modeling these would prevent simulation of large networks.

We implemented three overlay network designs: Pastry, Chord, and SkipNet. The Pastry implementation is described in Rowstron and Druschel [18]. Our Chord implementation is the one available on the MIT Chord web site [11], adapted to operate within our simulator. For our simulations, we run the Chord stabilization algorithm until no finger pointers need updating after all nodes have joined. We use two different implementations of SkipNet: a "basic" implementation based on the design in Section 3, and a "full" implementation that uses the enhancements described in Section 5. For "full" SkipNet, we run two rounds of stabilization for P-table entries before each experiment.

All our experiments were run both on a Mercator topology [22] and a GT-ITM topology [25]. The Mercator topology has $102,639$ nodes and $142,303$ links. Each node is assigned to one of $2,662$ Autonomous Systems (ASs). There are $4,851$ links between ASs in the topology. All figures shown in this section are for the Mercator topology. The experiments based on the Georgia Tech topology produced similar results.

7.1 Methodology

We measured the performance characteristics of lookups using the following evaluation criteria:

Relative Delay Penalty (RDP): The ratio of the latency of the overlay network path between two nodes to the latency of the IP-level path between them.

Physical network hops: The absolute length of the overlay path between two nodes, measured in IP-level hops.[1]

Number of failed lookups: The number of unsuccessful lookup requests in the presence of failures.

We also model the presence of organizations within the overlay network; each participating node belongs to a single organization. The number of organizations is a parameter to the experiment, as is the total number of nodes in the overlay. For each experiment, the total number of client lookups is twice the number of nodes in the overlay.

The format of the names of participating nodes is *org-name/node-name*. The format of data object names is *org-name/node-name/random-obj-name*. Therefore we assume that the "owner" of a particular data object will name it with the owner node's name followed by a node-local object name. In SkipNet, this results in a data object being placed on the owner's node; in Chord and Pastry, the object is placed on a node corresponding to the MD-5 hash of the object's name. For constrained load balancing experiments we use data object names that include the '*!*' delimiter following the name of the organization.

We model organization sizes two ways: a uniform model and a Zipf-like model. In the Zipf-like model, the size of an organization is determined according to a distribution governed by $x^{-1.25}+0.5$ and normalized to the total number of overlay nodes in the system. All other Zipf-like distributions mentioned in this section are defined in a similar manner.

We model three kinds of node locality: uniform, clustered, and Zipf-clustered. In the uniform model, nodes are uniformly spread throughout the overlay. In the clustered model, the nodes of an organization are uniformly spread throughout a single randomly chosen autonomous system. We ensure that the selected AS has at least $1/10$-th as many core router nodes as overlay nodes. For Zipf-clustered, we place organizations within ASes, as before. However, the nodes of an organization are spread throughout its AS as follows: A "root" physical node is randomly placed within the

[1] Mercator does not provide link latencies.

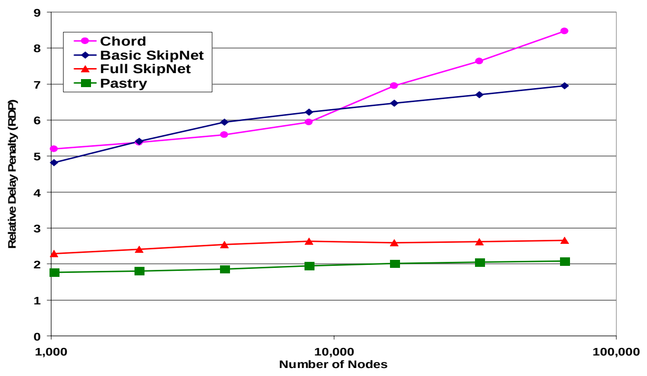

Figure 7. *RDP as a function of network size. Configuration: 1000 organizations with Zipf-like sizes, nodes and data names are Zipf-clustered.*

Chord	Basic SkipNet	Full SkipNet	Pastry
16.3	41.7	73.5	63.2

Table 1. *Average number of unique routing entries per node in an overlay with 2^{16} nodes.*

AS and all overlay nodes are placed relative to this root, at distances modeled by a Zipf-like distribution.

Data object names, and therefore data placement, is modelled similarly. In a uniform model, data names are generated by randomly selecting an organization and then a random node within that organization. In a clustered model, data names are generated by selecting an organization according to a Zipf-like distribution and then a random member node within that organization. For Zipf-clustered, data names are generated by randomly selecting an organization according to a Zipf-like distribution and then selecting a member node according to a Zipf-like distribution of its distance from the "root" node of the organization. Note that for Chord and Pastry, but not SkipNet, hashing spreads data objects uniformly among all overlay nodes in all these three models.

We model locality of data access by specifying what fraction of all data lookups will be forced to request data local to the requestor's organization. Finally, we model system behavior under Internet-like failures and study document availability within a disconnected organization. We simulate domain isolation by failing the links connecting the organization's AS to the rest of the network.

Each experiment is run ten different times, with different random seeds, and the average values are presented. For SkipNet, we used 128-bit random identifiers and a leaf set size of 16 nodes. For Pastry and Chord, we used their default configurations [11, 18].

7.2 Basic Routing Costs

To understand SkipNet's routing performance we simulated overlay networks with $N = 2^k$ nodes, where k ranges from 10 to 16. We ran experiments with 10, 100, and 1000 organizations and with all the permutations obtainable for organization size distribution, node placement, and data placement. The intent was to see how RDP behaves under various configurations. We were especially curious to see whether the non-uniform distribution of data object names would adversely affect the performance of SkipNet lookups, as compared to Chord and Pastry.

Figure 7 presents the RDPs measured for both implementations of SkipNet, as well as Chord and Pastry, for a configuration in which organization sizes, node placement, and data placement are all governed by Zipf-like distributions. Table 1 shows the average number of unique routing table entries per node in an overlay with 2^{16} nodes. All other configurations, including the completely uniform ones, exhibited similar results to those shown here.

Our conclusion is that basic SkipNet performs similarly to Chord and full SkipNet performs similarly to Pastry. This is not surprising since both basic SkipNet and Chord do not support network proximity-aware routing whereas full SkipNet and Pastry do. Since all our other configurations produced similar results, we conclude that SkipNet's performance is not adversely affected by non-uniform distributions of names.

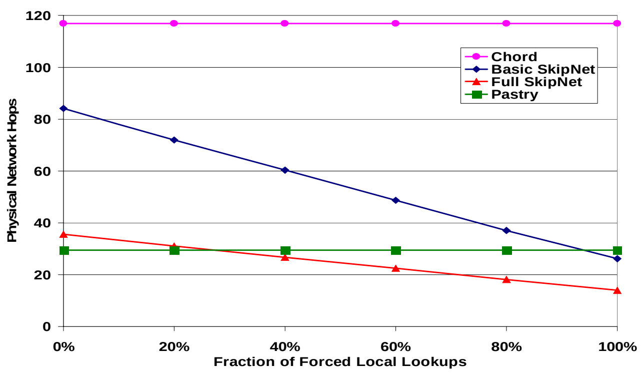

Figure 8. *Absolute lookup request latency as a function of data access locality (percentage of lookup requests forced to be within a single organization). Configuration: 2^{16} nodes, 100 organizations with Zipf-like sizes, nodes and data names are Zipf-clustered.*

7.3 Exploiting Locality of Placement

RDP only measures performance relative to IP-based routing. However, one of SkipNet's key benefits is that it enables localized placement of data. Figure 8 shows the average number of physical network hops for lookup requests for an experiment configuration containing 2^{16} overlay nodes and 100 organizations, with organization size, node placement, and data placement all governed by Zipf-like distributions. The x axis indicates what fraction of lookups were forced to be to local data (i.e. the data object names that were

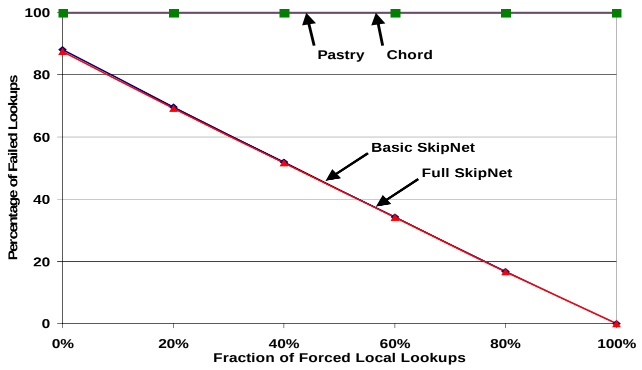

Figure 9. *Number of failed lookup requests as a function of data access locality (percentage of lookup requests forced to be within a single organization) for a disconnected organization. Configuration: 2^{16} nodes, 100 organizations with Zipf-like sizes, nodes and data names are Zipf-clustered.*

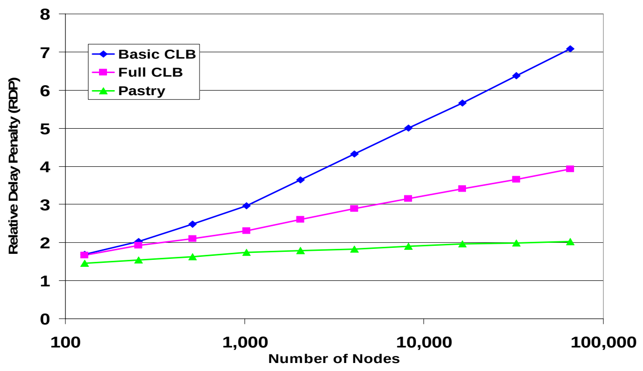

Figure 10. *RDP of lookups for data that is constrained load balanced (CLB) as a function of network size. Configuration: 100 organizations with Zipf-like sizes, nodes and data names are Zipf-clustered.*

looked up were from the same organization as the requesting client). The y axis shows the number of physical network hops for lookup requests.

As expected, both Chord and Pastry are oblivious to the locality of data references since they diffuse data throughout their overlay network. On the other hand, both versions of SkipNet show significant performance improvements as the locality of data references increases. It should be noted that Figure 8 actually understates the benefits gained by SkipNet because, in our Mercator topology, inter-domain links have the same cost as intra-domain links. In an equivalent experiment run on the GT-ITM topology, SkipNet end-to-end lookup latencies were over a factor of seven less than Pastry's for 100% local lookups.

7.4 Fault Tolerance

Locality of placement also improves fault tolerance. Figure 9 shows the number of lookup requests that failed when an organization was disconnected from the rest of the network.

This (common) Internet-like failure had catastrophic consequences for Chord and Pastry. The size of the isolated organization in this experiment was roughly 15% of the total nodes in the system. Consequently, Chord and Pastry will both place roughly 85% of the organization's data on nodes outside the organization. Furthermore, they must also attempt to route lookup requests with 85% of the overlay network's nodes effectively failed (from the disconnected organization's point-of-view). At this level of failures, routing is effectively impossible. The net result is a failed lookups ratio of very close to 100%.

In contrast, both versions of SkipNet do better the more locality of reference there is. When no lookups are forced to be local, SkipNet fails to access the 85% of data that is non-local to the organization. As the percentage of local lookups is increased to 100%, the percentage of failed lookups goes to 0.

7.5 Constrained Load Balancing

Figure 10 explores the routing performance of two different CLB configurations, and compares their performance with Pastry. For each system, all lookup traffic is organization-local data. The organization sizes as well as node and data placement are clustered with a Zipf-like distribution. The Basic CLB configuration uses only the numeric routing described in Section 3.5, whereas Full CLB makes use of the CLB proximity table described in Section 5.2.

The Full CLB curve shows a significant performance improvement over Basic CLB, justifying the cost of maintaining extra routing state. However, even with the additional routing table, the Full CLB performance trails Pastry's performance. The key observation, however, is that in order to mimic the CLB functionality with a traditional peer-to-peer overlay network, multiple routing tables are required, one for each domain that you want to load-balance across.

8 Conclusion

To become broadly acceptable application infrastructure, peer-to-peer systems need to address two basic requirements: the ability to control data locality and the ability to guarantee that routing paths remain local within an administrative domain whenever possible. To our knowledge, SkipNet is the first overlay network design that achieves both content and routing path locality, by clustering peers and organizing data according to their lexicographic addresses and names. At the same time, SkipNet has similar performance guarantees to other peer-to-peer systems, in that it requires logarithmic state per node and supports searches, insertions and deletions in logarithmic time.

SkipNet allows for both fine-grained and coarse-grained control over data placement. Content can be placed on a specific participating peer, by incorporating the peer's address into the content name. Similarly, SkipNet allows for content to be randomly and uniformly distributed within an arbitrary node naming subtree, as defined by a document's name.

The same lexicographic clustering of peers according to their addresses allows SkipNet to perform gracefully in the face of realistic Internet failures, such as loss of a domain's Internet connectivity.

Our evaluation efforts have demonstrated that SkipNet has similar performance behavior to other overlay networks such as Chord and Pastry. Even under skewed node and object placement distributions SkipNet maintains logarithmic performance behavior. Finally, our experiments have illustrated the benefits of content and routing path locality in SkipNet, as compared to systems like Chord and Pastry.

In future work, we plan to deploy a SkipNet across a testbed cluster of 2000 machines emulating a WAN in order to understand and analyze its behavior in the face of dynamic host joins and leaves and its flexibility as infrastructure for implementing a scalable event notification service [3].

References

[1] J. Aspnes and G. Shah. Skip graphs. Submitted for publication to SODA 2003.

[2] A. Brown and D. A. Patterson. Embracing Failure: A Case for Recovery-Oriented Computing (ROC). In *High Performance Transaction Processing Symposium*, Asilomar, CA, USA, October 2001.

[3] L. F. Cabrera, M. B. Jones, and M. Theimer. Herald: Achieving a global event notification service. In *HotOS VIII*, May 2001.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Security for peer-to-peer routing overlays. In *Proc. of the Fifth Symposium on Operating System Design and Implementation (OSDI)*. USENIX, December 2002.

[5] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. In *Microsoft Technical Report #MSR-TR-2002-82*, 2002. http://www.research.microsoft.com/∼antr/PAST/location.pdf.

[6] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, pages 1–12, Santa Clara, CA, June 2000. ACM.

[7] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[8] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, Sept. 1960.

[9] N. J. A. Harvey, J. Dunagan, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Microsoft Technical Report No. MSR-TR-2002-92, in preparation*, 2002.

[10] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized, peer-to-peer web cache. In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, July 2002.

[11] F. Kaashoek, R. Morris, F. Dabek, I. Stoica, E. Brunskill, D. Karger, R. Cox, and A. Muthitacharoen. The chord project, 2002.
http://www.pdos.lcs.mit.edu/chord/.

[12] P. Keleher, S. Bhattacharjee, and B. Silaghi. Are virtualized overlay networks too much of a good thing? In *First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002.

[13] C. Labovitz and A. Ahuja. Experimental Study of Internet Stability and Wide-Area Backbone Failures. In *Fault-Tolerant Computing Symposium (FTCS)*, June 1999.

[14] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. 9th ACM Symp. on Parallel Algorithms and Architectures*, pages 311–320, June 1997.

[15] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures*, pages 437–449, 1989.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, Aug. 2001.

[17] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proceedings of the Third International Workshop on Networked Group Communication*, Nov. 2001.

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

[19] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[20] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Third International Workshop on Networked Group Communications*, Nov. 2001.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[22] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin. The impact of routing policy on internet paths. In *INFOCOM*, pages 736–742, April 2001.

[23] M. Theimer and M. B. Jones. Overlook: Scalable name service on an overlay network. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, July 2002.

[24] A. Vahdat, J. Chase, R. Braynard, D. Kostic, and A. Rodriguez. Self-organizing subsets: From each according to his abilities, to each according to his needs. In *First International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002.

[25] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom '96*, April 1996.

[26] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.