Efficient Recovery From Organizational Disconnects in SkipNet

Nicholas J. A. Harvey, Michael B. Jones, Marvin Theimer, Alec Wolman

Abstract: *SkipNet is a scalable overlay network that provides controlled data placement and routing locality guarantees by organizing data primarily by lexicographic ordering of string names. A key side-effect of the SkipNet design is that all nodes from an organization form one or a few contiguous overlay segments. When an entire organization disconnects from the rest of the system, repair of only a few pointers quickly enables efficient routing throughout the disconnected organization; full repair is done as a subsequent background task. These same operations can be later used to efficiently reconnect an organization's SkipNet back into the global one.*

1 Introduction

SkipNet is a scalable, peer-to-peer overlay network that organizes nodes into a circular distributed data structure that concurrently supports two separate, but related address spaces. In one space, nodes belong to multiple rings where ring members are lexicographically ordered according to nodes' string names. In the other space, nodes are labeled with uniformly distributed numeric IDs. These numeric IDs define which rings a node belongs to in the first space. The combination of the two spaces enables SkipNet to provide efficient message routing as well as support several important locality properties.

Most notable of these properties is the ability to control the placement of data. SkipNet supports simultaneous use of multiple distributed hash tables (DHTs) that span varying subsets of the overlay nodes. In particular, DHTs can be defined over any set of nodes that share the same string name prefix. For example, if nodes belonging to the same organization all share an organization-specific name prefix then clients of the overlay can define DHTs whose data is load balanced across the nodes of a particular organization while still being accessible from any node in the overlay network.

SkipNet's design also allows it to guarantee that message routes traverse only intermediate nodes sharing the same name prefix as do the source and destination nodes. Thus local access to data stored within an organization can be obtained without having to worry about the associated message traffic having to traverse external nodes that might be either hostile or unavailable.

One of the more common forms of Internet failure is disconnection of an organization due to router misconfigurations and link and router faults [6, 7]. When such a disconnection occurs, SkipNet's locality properties enable a graceful degradation of functionality wherein local overlay traffic and hence access to data stored in locally defined DHTs still remains possible. Assuming that organizations assign node names with one or a few organizational prefixes, an organization's nodes are naturally arranged into one or a few contiguous overlay segments. Should an organization become disconnected, its segments remain internally well-connected and intra-segment traffic can be routed with the same $O(\log N)$ hop efficiency as before.

By forming only a few key routing pointers between the "edge" nodes of each segment, the entire organization can be connected into a separate SkipNet that can route traffic with similar efficiency: cross-segment traffic incurs an additional penalty that is proportional to the number of segments traversed. A background process repairs the additional routing pointers, thereby eliminating the cross-segment penalty. SkipNet's structure enables this repair process to be done in a manner that avoids unnecessary duplication of work. When the organization reconnects to the Internet, these same repair operations can be used to merge the organization's segments back into the global SkipNet.

In contrast, most previous scalable, peer-to-peer overlay designs [9, 10, 11, 13] place nodes in the overlay topology according to a unique random numeric ID only. Disconnection of an organization in most of these systems will result in its nodes fragmenting into many disjoint overlay pieces. During the time that these fragments are reforming into a single overlay, network routing efficiency may be poor or unbalanced, or may even fail.

SkipNet's basic design and performance are described in Section 2; a complete description can be found in in Harvey et al. [3, 4]. Section 3 describes the repair algorithms for disconnection and reconnection, and presents some performance evaluation results. Section 4 concludes the paper.

2 SkipNet Overview

SkipNet is a scalable peer-to-peer overlay network designed to support two key locality properties: *content locality* and *path locality*. These locality properties address two notable disadvantages of many existing overlay designs: it is difficult to control where data is stored and it is difficult to guarantee that routing paths remain within an administrative domain. Content locality is the ability to place data either on specific overlay nodes or to load balance it across the nodes within a specific organization. Path locality is the ability to guarantee that when two overlay nodes within the same organization communicate, any intermediate routing hops also remain within that same organization.

Content and path locality provide a number of key benefits with respect to availability, performance, manageability, and security. As we and others [4, 5, 12] have noted, much of the data stored within peer-to-peer systems can still be expected to exhibit significant locality in clients' access patterns. Thus, for example, organizations may wish to make data globally available while still storing it locally in order to obtain better intra-organizational availability and performance. Content and path locality support this style of non-uniform peer-to-peer system. In particular, they also

[1]Authors' address: Microsoft Research, Microsoft Corporation, Redmond, WA, 98052, {nickhar, mbj, theimer, alecw}@microsoft.com

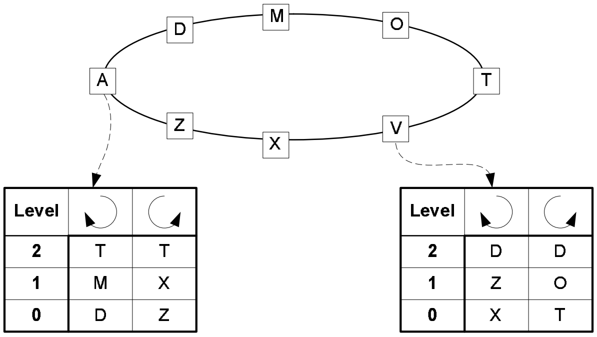

Figure 1: *SkipNet nodes ordered by name ID. Routing tables of nodes A and V are shown.*

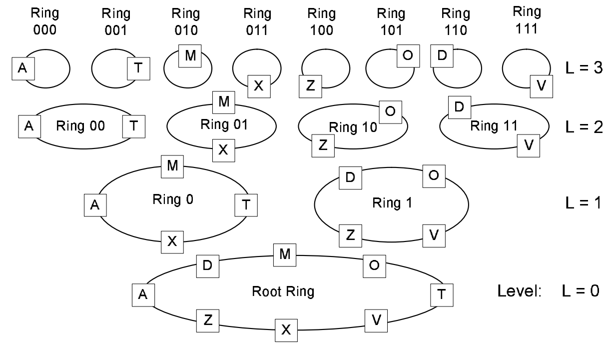

Figure 2: *The full SkipNet routing infrastructure for an 8 node system, including the ring labels.*

enable an organization to become disconnected from the Internet while still allowing overlay members of the organization to communicate with each other and access data stored within the organization. Explicit content placement onto a single node or across a well-defined set of nodes also enhances overall system manageability because the relevant nodes can be provisioned to support the anticipated access frequency for the set of data items being stored. Finally, path locality provides significant security benefits since potentially malicious routers in other administrative domains cannot affect intra-domain traffic.

2.1 The Basic SkipNet Structure

SkipNet's basic design is derived from ideas underlying the in-memory Skip List data structure [8]. 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 *name IDs* 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 2.2.

In the basic SkipNet design each node stores $2 \log N$ pointers, where N is the number of nodes in the overlay system. Each node's set of pointers is called its *routing table*, or R-Table, since the pointers are used to route message traffic between nodes. The pointers at level h of a given node's routing table point to nodes that are roughly 2^h nodes to the left and right of the given node.

Figure 1 depicts a SkipNet containing eight nodes and shows the routing table pointers that nodes A and V maintain. The SkipNet in Figure 1 is a "perfect" SkipNet: each level h pointer traverses exactly 2^h nodes. Figure 2 depicts the same SkipNet of Figure 1, arranged to show all node interconnections at every level simultaneously. All nodes are connected by the *root ring* formed by each node's pointers at level 0. The pointers at level 1 point to nodes that are 2 nodes away and hence the overlay nodes are implicitly divided into two disjoint rings. Similarly, pointers at level 2 form four 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.

Maintaining a perfect SkipNet in the presence of insertions and deletions is very expensive. To facilitate efficient insertions and deletions, we derive a probabilistic SkipNet design. Each ring at level h is split into two rings at level $h+1$ by having each node randomly and uniformly choose to which of the two rings it belongs. With this probabilistic scheme, insertion/deletion of a node only affects two other nodes in each ring to which the node has randomly chosen to belong. Furthermore, a pointer at level h still skips over 2^h nodes in expectation, and routing is possible in $O(\log N)$ forwarding hops with high probability.

Each node's random choice of ring memberships can be encoded as a unique binary number, which we refer to as the node's *numeric ID*. As illustrated in Figure 2, the first h bits of the number determine ring membership at level h. For example, node X's numeric ID is 011 and its membership at level 2 is determined by taking the first 2 bits of 011, which designate Ring 01. One way to obtain a unique, random numeric ID is by using a collision-resistant hash (such as SHA-1) of the node's DNS name.

Because the numeric IDs of nodes are unique they can be thought of as a second address space that is maintained by the same SkipNet data structure. Whereas SkipNet's string address space is populated by node name IDs that are *not* uniformly distributed throughout the space, SkipNet's numeric address space is populated by node numeric IDs that *are* uniformly distributed. The uniform distribution of numeric IDs in the numeric space is what ensures that our routing table construction yields routing table entries that skip over the appropriate number of nodes.

2.2 Routing by Name ID

Routing/searching by name ID in SkipNet is based on the same basic principle as searching in Skip Lists: Follow pointers that route closest to the intended destination. At each node, a 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 name ID is closest to the destination.

Since nodes are ordered by name ID along each ring and a message is never forwarded past its destination, all nodes encountered during routing have name IDs between the source and the destination. Thus, when a message originates at a node whose name ID shares a common prefix with the destination, all nodes traversed by the message have name IDs that share that same prefix. Rings are doubly-linked so that routing can use either right or left pointers depending upon whether the source's name ID is smaller or greater than the destination's.

The number of message hops when routing by name ID is $O(\log N)$ with high probability. For a proof, see [3].

2.3 Routing by Numeric ID

It is also possible to route messages efficiently to a given numeric ID. In brief, the routing operation begins by examining nodes in the level 0 ring until a node is found whose numeric ID matches the destination numeric ID in the first digit. At this point the routing operation jumps up to this node's level 1 ring, which must also contain the destination node. The routing operation then examines nodes in this level 1 ring until a node is found whose numeric ID matches the destination numeric ID in the second digit. As before, we know that this node's level 2 ring must also contain the destination node, and thus the routing operation proceeds in this level 2 ring.

This procedure repeats until we cannot make any more progress — we have reached a ring at some level h such that none of the nodes in that ring share $h + 1$ digits with the destination numeric ID. We must now deterministically choose one of the nodes in this ring to be the destination node. Our algorithm defines the destination node to be the node whose numeric ID is numerically closest to destination numeric ID amongst all nodes in this highest ring.

As an example, imagine that the numeric IDs in Figure 2 are 4 bits long and that node Z's ID is 1000 and node O's ID is 1001. If we want to route a message from node A to destination 1011 then A will first forward the message to node D because D is in ring 1. D will then forward the message to node O because O is in ring 10. O will forward the message to Z because it is not in ring 101. Z will forward the message onward around the ring (and hence back) to O for the same reason. Since none of the members of ring 10 belong to ring 101, node O will be picked as the final message destination because its numeric ID is closest to 1011 of all ring 10 members.

The number of message hops when routing by numeric ID is $O(\log N)$ with high probability. For a proof, see [3].

2.4 Constrained Load Balancing

One of the most interesting capabilities that SkipNet supports is *constrained load balancing* (CLB). This is the ability to concurrently support multiple DHTs, each of which may span a client-specified set of nodes that all share the same name prefix. CLB is possible because SkipNet maintains two separate, but related address spaces, one of which supports efficient range queries over node name IDs. To implement CLB, we divide a data object's name into two parts: a part that specifies the set of nodes over which DHT load balancing should be performed (the *CLB domain*) and a part that is used as input to the DHT's hash function (the *CLB suffix*). In SkipNet the special character '!' is used as a delimiter between the two parts of the name.

For example, suppose we stored a document using the name *msn.com/DataCenter!TopStories.html*. The CLB domain indicates that load balancing should occur over all nodes whose names begin with the prefix *msn.com/DataCenter*. The CLB suffix, *TopStories.html*, is used as input to the DHT hash function, and this determines the specific node within *msn.com/DataCenter* on which the document will be placed.

To search for a data object that has been stored using CLB, we first search for any node within the CLB domain using search by name ID. To find the specific node within the domain that stores the data object, we perform a search by numeric ID within the CLB domain for the hash of the CLB suffix.

The search by name ID is unmodified from the description in Section 2.2, and takes $O(\log N)$ message hops. The search by numeric ID is constrained by a name ID prefix and thus at any level must effectively step through a doubly-linked list rather than a ring. Upon encountering the right boundary of the list (as determined by the name ID prefix boundary), the search must reverse direction in order to ensure that no node is overlooked. Reversing directions in this manner affects the performance of the search by numeric ID by at most a factor of two, and thus $O(\log N)$ message hops are required in total.

2.5 Enhancements

Links in the routing structure shown in Figure 2 are determined strictly by nodes' name IDs and numeric IDs, which means that the SkipNet overlay is constructed without consideration of the physical network topology. To improve routing performance, SkipNet maintains two additional *proximity* routing tables, one per address space, that provide pointers that reflect network proximity of nodes much the way that Pastry's proximity-aware routing tables do [2]. Further details can be found in both [4] and [3].

An important point to understand is that both proximity tables are used only to optimize routing efficiency. Message routing reverts to using an R-Table pointer whenever use of a proximity table pointer would violate path locality or the relevant proximity table pointer is invalid (e.g. due to organizational disconnect).

To maintain the root ring correctly, each SkipNet node also maintains a *leaf set* that points to additional nodes along the root ring, for redundancy. In our current implementation we use a leaf set size of 16, just as Pastry does. Whereas root ring pointers are monitored and repaired in an expedited manner, all other SkipNet state is updated and repaired in a background fashion.

3 Failure Recovery Algorithms

When an organization is disconnected from the Internet, its nodes will be able to communicate with each other over IP but will not be able to communicate with nodes outside the organization. If the organization's nodes' names employ only a few organizational prefixes then the nodes are mostly contiguous in SkipNet, and hence the global SkipNet will partition itself into several disjoint, but internally well-connected, segments. This is illustrated in Figure 3.

Because of SkipNet's path locality property, message traffic within each segment will be unaffected by discon-

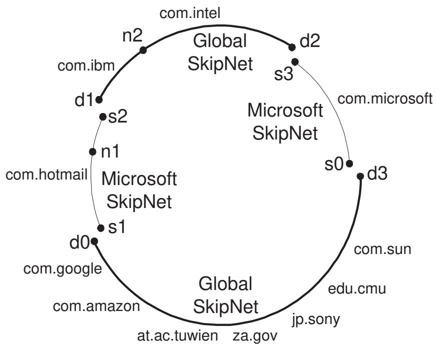

Figure 3: *Two partitioned SkipNets to be merged.*

nection and will continue to be routed with $O(\log M)$ efficiency, where M is the number of nodes within the segment. Assuming that the disconnecting organization constitutes a small fraction of the global SkipNet, cross-segment traffic among the global portions of the SkipNet will also remain largely unaffected because most cross-segment pointers among global segments will remain valid. This will not be true for the segments of the disconnected organization.

Gracefully handling a partition in the underlying IP network has two aspects: continuing to provide internal connectivity for the duration of the partition, and efficiently repairing the overlay when the underlying IP network partition heals. Maintaining internal connectivity of the overlay requires that communications be possible both within each overlay segment and across segments that still have IP connectivity to each other. Repairing the overlay when the partition heals involves reestablishing communications between overlay segments that were formerly unreachable by IP. Thus, the primary repair task after both disconnection and reconnection is the merging of overlay segments.

The algorithms employed in both the disconnection and reconnection cases are very similar: SkipNet segments must discover each other and then be merged together. For the disconnect case, the organization segments are merged into a separate SkipNet and the global segments are merged to reform the global SkipNet. For the reconnect case, all segments from the two separate SkipNets are merged into a single SkipNet.

3.1 Discovery Techniques

When an organization disconnects from the Internet there is no guarantee that the resulting non-contiguous segments will have pointers into each other. Therefore its segments may not be able to find each other using only SkipNet pointers. To solve this discovery problem we assume that organizations will divide their nodes into a relatively small number of name segments and that they designate some number of nodes in each segment as "well-known". For instance, Microsoft might maintain well-known members of segments with name prefixes *microsoft.com*, *hotmail.com*, *xbox.jp*, etc. Each node in an organization maintains a list of these well-known nodes and uses them as contact points between the various overlay segments.

When an organization reconnects to the Internet, the organizational and global SkipNets discover each other through their segment edge nodes. Since each node main-

```
ConnectRootLevel(n1, n2) {
  edgeNodes = GatherEdgeNodeInfo(n1, n2, null)
  Connect edge node pairs.
}

GatherEdgeNodeInfo(n1, n2, msg) {
  n2 routes msg to n1 in its SkipNet.
  Msg will arrive at d1.
  d1 appends d1 and next neighbor, d0, to msg contents.
  d1 sends msg directly to n1 over IP.
  n1 routes msg to d0 in its SkipNet.
  Msg will arrive at s1.
  if (memberOf(s0, msg contents))  // => all segments
    return msg contents            //    traversed
  else  // => Message needs to discover more edge nodes
    s1 appends s1 and next neighbor, s0, to msg contents.
    return GatherEdgeNodeInfo(s0, d0, msg)
}
```

Figure 4: *SkipNet root ring connection algorithm.*

tains a leaf set, if a node discovers that one side of its leaf set, but not the other, is completely unreachable then it concludes that a disconnect event has occurred and that it is an edge node of a segment. These edge nodes keep track of their unreachable leaf set pointers and periodically ping them for reachability; should a pointer become reachable, the node initiates the merge process. Note that merging two previously independent SkipNets together—for example, when a new organization joins the system—is functionally equivalent to reconnecting a previously connected one, except that an alternate means of discovery is needed.

3.2 Connecting Root Ring Segments

The segment merge process is comprised of two steps: repair of the root ring pointers and repair of the pointers for all higher-level rings. The first step can be done quickly, as it only involves repair of the root ring pointers of the edge nodes of each segment. Once the first step has been done it will be possible to route messages correctly among nodes in different segments and to do so with $O(S \log M)$ efficiency, where S is the total number of segments and M is the maximum number of nodes within a segment. As a consequence, the second, more expensive step can be done as a background task, as described in Section 3.3.

The key idea for connecting SkipNet root ring segments is to discover the relevant edge nodes by having a node in one segment route a message towards the name ID of a node in the other segment. This message will be routed to the edge node in the first segment that is lexicographically nearest to the other node's name ID. By repeating this process one can enumerate all edge nodes and hence all segments.

The actual inter-segment pointer updates are then done as a single atomic operation among the segment edge nodes, using distributed two-phase commit. This avoids routing inconsistencies where a message destined for a specific node on one segment inadvertently ends up at a different node in another overlay segment because the segments to be merged do not yet form a fully connected root ring.

To illustrate, Figure 3 shows two SkipNets to be merged, a Microsoft SkipNet and a global SkipNet, each containing two different name segments. Suppose that node $n1$ knows of node $n2$'s existence. Node $n1$ will send a message to node $n2$ (over IP) asking it to route a search message towards $n1$ in the global SkipNet. $n2$'s message will end up at node $d1$ and, furthermore, $d1$'s neighbor on the global

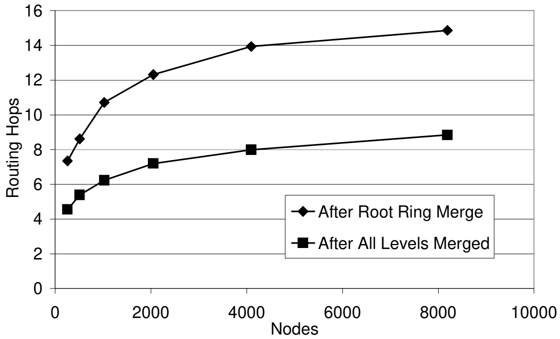

Figure 5: *Number of routing hops taken to route inter-organizational messages, as a function of network size, after an organization's internal SkipNet has been reconnected to the global SkipNet root ring and after the merge has been fully completed.*

SkipNet will be $d0$. $d1$ sends a reply to $n1$ (over IP) telling it about $d0$ and $d1$. $n1$ routes a search message towards $d0$ on the Microsoft SkipNet to discover $s1$ and $s0$ in the same manner. The procedure is iteratively invoked using $s0$ and $d0$ to gain information about $s2$, $s3$, $d2$, and $d3$. Figure 4 presents the algorithm in pseudo-code.

Immediately following root ring connection, messages sent to cross-segment destinations will be routed efficiently. Cross-segment messages will be routed to the edge of each segment they traverse and will then hop to the next segment using the root ring pointer connecting the segments. This leads to $O(S \log M)$ routing efficiency. When an organization reconnects its fully repaired SkipNet root ring to the global one, traffic destined for nodes external to the organization will be routed in $O(\log M)$ hops to an edge node of the organization's SkipNet. The root ring pointer connecting the two SkipNets will be traversed and then $O(\log N)$ hops will be needed to route traffic within the global SkipNet. Note that traffic that does not have to cross between the two SkipNets will not incur this routing penalty.

To experimentally confirm the behavior of SkipNet's disconnection and merge algorithms we implemented them and then ran them in an extended version of the packet-level discrete event simulator available from [10]. The simulator was extended to support disconnection of AS subnetworks. The details of our experiments and experimental setup are described in [3]. Figure 5 shows the routing performance we observed between a previously disconnected organization and the rest of the system once the organization's SkipNet root ring has been connected to the global SkipNet root ring. We also show the routing performance observed when all higher level pointers have been repaired.

3.3 Repairing Routing Pointers following Root Ring Connection

Once the root ring connection phase has completed we can update all remaining pointers that need repair using a background task. We present here an algorithm for doing this that avoids unnecessary duplication of work through appropriate ordering of repair activities.

The key idea is that we recursively repair pointers at one level by using correct pointers at the level below to find the desired nodes in each segment. All pointers at one level must be repaired across a segment boundary before repair of a higher level can be initiated. To illustrate, consider Figure 6, which depicts a single boundary between two SkipNet segments after pointers have been repaired. Figure 7 presents an algorithm in pseudo-code for repairing pointers above the root ring across a single boundary. We begin by discussing the single boundary case, and later we extend our algorithm to handle the multiple boundary case.

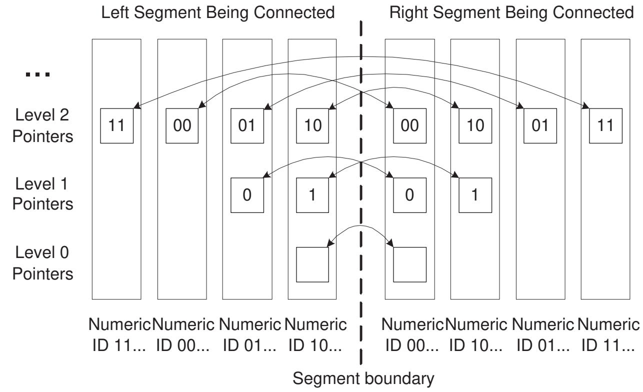

Figure 6: *Nodes whose pointers have been repaired at the boundary of two SkipNet segments.*

```
// Called initially with level h=0 at node
// to the left of the merge point
PostMergeRepair(h) {
  Find closest node to left whose numeric ID matches
      mine in the first h bits and whose ID differs from
      mine in the next bit, by following level h
      pointers to the left.
  On my node:
    cont = FixMyRightPointer(h+1)
    if (cont) PostMergeRepair(h+1)
  In parallel, on the other node:
    cont2 = FixMyRightPointer(h+1)
    if (cont) PostMergeRepair(h+1)
}

FixMyRightPointer(h) {
  Search right using level h-1 pointers until a node is
      found that matches my numeric id in h bits.
  Connect our level h pointers.
  if (pointers are already equal)
    return false
  else
    return true
}
```

Figure 7: *Level h ring repair algorithm for a single inter-segment boundary.*

Assume that the root ring pointers have already been correctly connected. There are two sets of two pointers to connect between the segments at level 1: the ones for the routing ring labeled 0 and the ones for the routing ring labeled 1 (see Figure 2). We can repair the level 1 ring labeled 0 by traversing the root (level 0) ring from one of the edge nodes until we find nodes in each segment belonging to the ring labeled 0. The same procedure is followed to correctly connect the level 1 ring labeled 1. After the level 1 rings, we use the same approach to repair the four level 2 rings.

Because rings at higher levels are nested within rings at lower levels, repair of a ring at level $h + 1$ can be initiated by one of the nodes that had its pointer repaired for the enclosing ring at level h. A repair at level $h+1$ is unnecessary if the level h ring (a) contains only a single member or (b) does not have an inter-segment pointer that required repair. The latter termination condition implies that most rings—

and hence most nodes—in the global SkipNet will not, in fact, need to be examined for potential repair.

The total work involved in this repair algorithm is $O(M \log(N/M))$, where M is the size of the disconnecting/reconnecting SkipNet segment and N is the size of the external SkipNet. Note that rings at level $h+1$ can be repaired in parallel once their enclosing rings at level h have been repaired across all segment boundaries. Thus, the repair process for a given segment boundary parallelizes to the extent supported by the underlying network infrastructure. The analysis behind these claims is omitted for space reasons and can be found in [3].

To repair multiple segment boundaries, we simply call the algorithm described above once for each segment boundary. In the current implementation, we perform this process iteratively, waiting for the repair operation to complete on one boundary before initiating the repair at the next boundary. In future work, we plan to investigate initiating the segment repair operations in parallel — the open question is how to avoid repair operations from different boundaries interfering with each other.

3.4 Repairing Proximity Table Entries

In normal operation, the routing table entries in both of a node's proximity routing tables are updated periodically using information gathered from the node's R-Table. Once the R-Table repair algorithms above have run then these periodic updates will likewise repair the node's proximity tables with no resulting increase in maintenance traffic.

4 Conclusion

Real-world data access patterns exhibit locality and hence we argue that peer-to-peer overlay networks should provide support for both content and path locality. One common form of locality that we expect is placement of globally accessible data within the organizations that own that data. We also expect that organizations will tend to access their own data more heavily than that of other organizations. Consequently there is value in enabling an organization's overlay network to continue functioning internally even when its member nodes have become partitioned from the rest of the overlay network.

SkipNet supports content and path locality by means of two separate, but related address spaces, one of which orders nodes lexicographically by their string names. The combination of the two spaces enables the definition of multiple globally accessible DHTs whose storage scopes span subsets of all overlay nodes. Assuming that organizations assign their nodes' names with a small number of unique organizational prefixes, they can thus store data locally while still making it globally available.

One of the more common forms of Internet failure is disconnection of an organization. The primary contribution of this paper is a description of how SkipNet enables efficient recovery from these failures. Because of the assumption that organizations' nodes will share one or a few name prefixes, a disconnect will result in a small number of internally well-connected overlay segments, each of which is able to route internal messages efficiently. Only a few routing pointers need to be repaired to connect segments into a functioning overlay network able to route with mildly degraded efficiency throughout the organization. Similarly, once a network partition has healed, the same approach can be used to reconnect the organization's SkipNet back into the global one.

Because efficient routing is quickly obtained, full repair of all nodes' routing tables can be done as a background task for both disconnection and reconnection. SkipNet's structure enables these repairs to be ordered in a manner that avoids unnecessary duplication of work.

Several important issues remain. Most notably, we have not yet explored how to initiate and perform multiple segment repair operations in parallel. Another problem is that segment edge nodes can become routing hot spots for both successful, as well as failed, cross-segment traffic. Finally, we also plan to find out how SkipNet behaves in practice by using it as the underpinning for the Herald global event notification service [1], which is currently being built to run on a large test bed cluster of machines.

References

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

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

[3] 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. Technical Report MSR-TR-2002-92, Microsoft Research, 2002.

[4] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proceedings of 4th USITS*, Mar. 2003. http://research.microsoft.com/sn/Herald/papers/usits_abstract.html.

[5] 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.

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

[7] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? In *Proceedings of 4th USITS*, Mar. 2003.

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

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM '01 Conference*, August 2001.

[10] 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)*, pages 329–350, Nov. 2001.

[11] 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*, pages 149–160, August 2001.

[12] 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.

[13] 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.