Appears in *Proceedings of the 11th IEEE International Conference on Network Protocols* (ICNP 2003)

Exploiting Routing Redundancy via Structured Peer-to-Peer Overlays

Ben Y. Zhao, Ling Huang, Jeremy Stribling,
Anthony D. Joseph, and John D. Kubiatowicz
Computer Science Division, U. C. Berkeley
{ravenben, hling, strib, adj, kubitron}@cs.berkeley.edu

Abstract

Structured peer-to-peer overlays provide a natural infrastructure for resilient routing via efficient fault detection and precomputation of backup paths. These overlays can respond to faults in a few hundred milliseconds by rapidly shifting between alternate routes. In this paper, we present two adaptive mechanisms for structured overlays and illustrate their operation in the context of Tapestry, a fault-resilient overlay from Berkeley. We also describe a transparent, protocol-independent traffic redirection mechanism that tunnels legacy application traffic through overlays. Our measurements of a Tapestry prototype show it to be a highly responsive routing service, effective at circumventing a range of failures while incurring reasonable cost in maintenance bandwidth and additional routing latency.

1. Introduction

With the continued growth of the Internet, developers are deploying new and larger scale network applications, such as file sharing, instant messaging, streaming multimedia and voice-over-IP (VoIP). These applications are placing increasingly heavy demands on the Internet infrastructure, requiring highly reliable delivery and quick adaptation in the face of failure.

Unfortunately, it is becoming increasingly difficult to meet these criteria. The growing size and complexity of the network lead to frequent periods of wide-area disconnection or high packet loss. A variety of factors contribute to this, including router reboots, maintenance schedules, BGP misconfigurations, cut fibers and other hardware faults. The resulting loss and jitter on application traffic creates significant roadblocks to the widespread deployment of "realtime" applications such as VoIP.

The magnitude of this loss and jitter is a function of the routing protocol's response time to faults, including time to detect a fault, construct a new path, and reroute traffic. Recent work has analyzed the fault-recovery time of intra-AS protocols such as IS-IS for large IP backbone networks [11]. It found that while overall recovery is on the order of five or six seconds, the majority of delay is not due to fault-detection or path recalculation; it arises from timed delay between fault-detection and update of routing entries in the linecards. The latter is exacerbated by hardware features of current routers. Without these factors, it is reasonable to expect IS-IS to respond to route failures in two or three seconds.

Wide-area route convergence on BGP [23] is significantly slower. Recent work has identified interactions between protocol timers as the fundamental cause of delayed convergence. Because BGP disseminates reachability updates hop by hop between neighbors, full propagation across a network can take $30n$ seconds, where n is the longest alternative path between a source and any destination AS, and 30 is the length of a typical BGP rate limiting timer [16]. Unfortunately, studies have shown a significant growth in BGP routing tables fueled by stub ASes [2], meaning the delayed convergence problem will only grow in severity with time.

One commonality between deployed protocols is that the network is treated as an unstructured graph with arbitrary connections, implying the potential for any-to-any dependencies between peers. Local changes must therefore be propagated to all other peers in the network. Attempts to aggregate such state to reduce bandwidth is a primary motivation for several of the timers that contribute to the route convergence delay. Addressing schemes such as CIDR [22] that introduce hierarchy and structure into the namespace reduce the amount of routing state, and potentially reduce the need for long term timers. The problem remains that inter-AS routing is driven by peering agreements and policy, making state reduction a difficult problem.

In this paper, we make two contributions to address these issues. First, we propose the use of structured peer-to-peer (P2P) overlay networks [6] as resilient routing infrastruc-

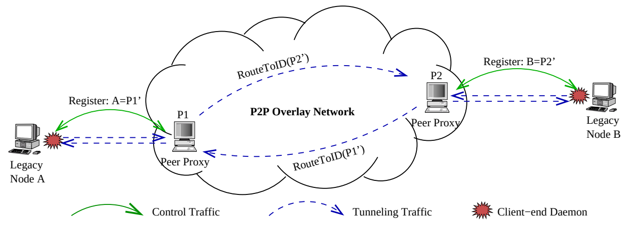

Figure 1. ***Tunneling traffic through a wide-area overlay.*** **Legacy application nodes tunnel wide-area traffic through the overlay.**

tures. Since they support efficient fault detection and precomputation of backup paths, P2P overlays can rapidly switch between alternate paths during network failure. Second, we describe a general redirection and addressing mechanism that transparently redirects IP traffic from legacy applications through the overlay—providing stable communication to legacy applications in the face of a variety of faults. Figure 1 illustrates this high level architecture.

Structured P2P overlays utilize local routing resources of $O(Log(N))$, resulting in low control traffic and fast propagation of route updates. In contrast, previous unstructured approaches, like Resilient Overlay Networks [1], exhibit the same state management problem as seen in BGP. Consequently, unstructured networks incur $O(N^2)$ total beacon messages for fault detection and backup path construction, preventing the system from scaling to hundreds of nodes.

Two adaptive routing mechanisms that are straightforward to implement in structured overlays are *First Reachable Link* (FRLS) and *constrained multicast*. The first chooses between one of a set of routing paths that are most stable, while the second sends duplicate packets along alternate paths to adapt to high packet loss. To maintain their level of routing redundancy, structured P2P networks must continually refresh their routing tables. The result is a decoupling of the discovery of backup paths ("precomputation") from rapid adaptation to failure ("route selection"). Precomputation shields applications from slower route convergence at the network layer.

In the following, we demonstrate the benefits of overlay routing through structured P2P overlays using a combination of analysis, simulation, and experimental measurements. We deploy a prototype of the Tapestry system [10, 32], a fault-resilient overlay that implements these adaptive mechanisms. We show that Tapestry can recover from network failures in under 700 milliseconds while using less than 7 Kilobytes/second of per-node beaconing traffic—agile enough to support most streaming multimedia applications. Our design should be scalable and easy for Internet Service Providers (ISPs) to deploy, providing fault-resilient routing services to legacy applications.

The remainder of this paper is as follows: Section 2 describes how to utilize a structured peer-to-peer overlay network to quickly detect routing failures and route around them using precomputed backup paths. Then, Section 3 describes the architecture of a fault-tolerant traffic tunneling service built with such an overlay. We evaluate of our design in Section 4. Finally, we discuss related work in Section 5 and conclude in Section 6.

2. Fault Tolerant Overlay Routing

In this section, we examine the fault-tolerant routing properties of structured peer-to-peer overlay networks. First, we give an overview of these overlays and their generalized properties. Our algorithms require only the basic key-to-node mapping function common to all of these protocols. While we motivate our examples and perform measurements using a locally designed protocol (Tapestry), our results should extend to others. We then discuss mechanisms for efficient fault detection. Finally, we propose techniques for routing around link failures and loss, and for maintaining routing redundancy across failures.

2.1. Structured Peer-to-Peer Overlays

Structured peer-to-peer (P2P) overlay networks have recently gained popularity as a platform for the construction of resilient, large-scale distributed systems [9, 18, 19, 21, 25, 27, 32]. Structured overlays conform to a specific graph structure that allows them to locate objects by exchanging $O(\log N)$ messages in an overlay of N nodes.

A *node* represents an instance of a participant in the overlay (one or more nodes may be hosted by a single physical IP host). Participating nodes are assigned *nodeIDs* uniformly at random from a large *identifier space*. Application-specific objects are assigned unique identifiers called *keys*, selected from the same identifier space. For example, Pastry [25], Tapestry [10, 32], Chord [27], Kademlia [19] and Skipnet [9] use an identifier space of n-bit integers modulo 2^n ($n = 160$ for Chord, Kademlia, Skipnet and Tapestry, $n = 128$ for Pastry).

Overlays dynamically map each key to a unique live node, called its *root*. These overlays support routing of messages with a given key to its root node, called *Key-Based Routing* [6]. To deliver messages efficiently, each node maintains a *routing table* consisting of the nodeIDs and IP addresses of the nodes to which the local node maintains overlay links. Messages are forwarded across overlay links to nodes whose nodeIDs are progressively closer to the key in the identifier space, such as in Figure 2. Each system defines a function that maps keys to nodes. For example, Tapestry maps a key to the live node whose nodeID has the longest prefix match, where the node with the next higher

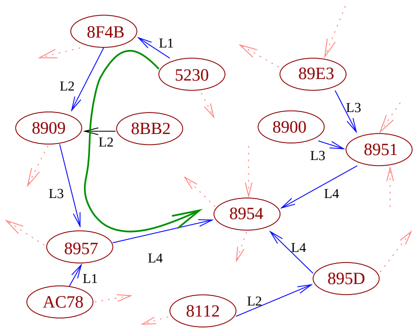

Figure 2. ***Routing example in Tapestry.* Routing path taken by a message from node `5230` towards node `8954` in Tapestry using hexadecimal digits of length four. As with other key-based routing (KBR) overlays, each hop resolves one digit.**

nodeID value is chosen for a digit that cannot be matched exactly.

An important benefit of Key-Based Routing (KBR) is that any node satisfying the namespace constraints can serve as a next routing hop. For example, in Tapestry or Pastry, the first hop of a message routing to the key 1111 requires only that the node's nodeID begins with 1. This property allows each overlay node to proactively maintain a small number of backup routes in its routing table. Upon detecting a failed outgoing link, a router can rapidly switch to a backup link, providing *fast failover*. In the background, the overlay networking algorithms can adapt to failure by restoring (*repairing*) the redundancy in backup links. We discuss this further in Section 2.3.

Most structured P2P protocols support Key-Based Routing, but differ in the performance tradeoffs they make. Where appropriate, we will use details of Tapestry to illustrate our points in later discussions. Tapestry [32] is a structured peer-to-peer overlay that uses prefix matching to route messages to keys, where each additional hop matches the key by one or more digits. For each routing entry, Tapestry tries to locate the nearest node with the required prefix in its nodeID using a nearest neighbor search algorithm [10].

One discerning factor is *proximity routing*, the latency optimization of routes using knowledge of physical network latencies during overlay construction to improve end-to-end latency. The overhead of routing on an overlay is generally measured as the Relative Delay Penalty (RDP), the ratio of overlay routing latency to IP latency. As we will show in Section 4, proximity enabled overlays such as Tapestry provide low overhead over IP. Thus, tunneled IP traffic gains resilience to faults without unreasonable increases in delay.

2.2. Efficient Fault Detection

Over 70% of Internet failures have durations less than two minutes [7], making fast response time the key objective for any fault-resilient routing mechanism. Traditionally,

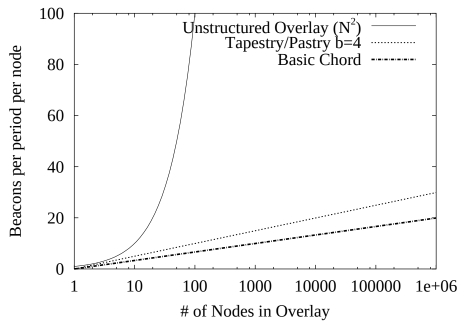

Figure 3. ***Fault-detection Bandwidth.* Unstructured overlay networks consume far more maintenance bandwidth than structured P2P networks. Bandwidth here is measured in beacons per node per beacon period.**

response time (T_r) is the sum of fault detection time (T_f) and path discovery time (T_p): $T_r = T_f + T_p$. Proactively maintaining backup paths allows us to immediately redirect traffic after failure detection, eliminating path discovery time ($T_r \approx T_f$). We now focus on minimizing T_f.

Application-level protocols generally depend on soft-state beacons (or heartbeat messages) to detect link and node failures. Bandwidth (B) is often the limiting factor, and is proportional to the product of the number of entries in each routing table (E) and the heartbeat frequency (F): $B \propto E \cdot F$. Nodes in structured peer-to-peer overlays maintain routing state (E) that grows logarithmically to the network size (N): $E \propto \log(N)$. Compared to unstructured protocols [1] with linear growth routing state ($E \propto N$), these overlays can send beacons at significantly higher frequencies while consuming identical bandwidth. Figure 3 shows the number of heartbeats sent per period for both unstructured overlays such as RON and structured P2P overlays such as Tapestry and Chord.

Total Bandwidth Consumption: The number of beacons sent per period is useful, but does not capture their true impact on the network. Since queuing delay and congestion happen on a per IP router basis, identical messages traversing different overlay hops can place different stresses on the network depending on the number of IP hops traversed. A more accurate measure is the *Total Bandwidth Consumption* (TBC), measured as a bandwidth distance product:

$$TBC = (msgs/sec) \cdot (bytes/msg) \cdot IPHops \qquad (1)$$

This metric reflects the fact that longer paths have a greater opportunity cost since they consume more total resources. Crossing fewer IP hops also means routes with low TBC have fewer chances of encountering failures.

Messages routing across latency-optimized overlays cross a smaller number of IP routers and incur a lower TBC. We further quantify this effect in Figure 10 in Section 4.1, by comparing simulated TBC for a single struc-

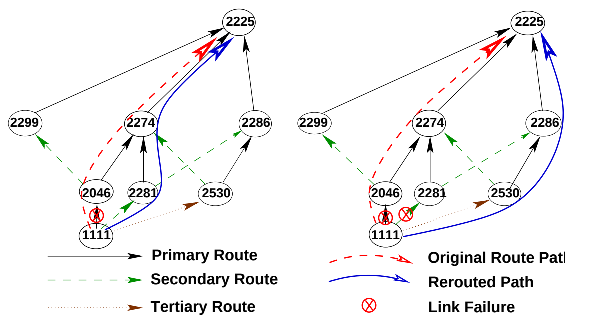

Figure 4. ***First Reachable Link.*** **Using simple route selection (First Reachable Link or FRLS) to circumvent single and multiple failed links on an overlay path from** `5230` **to** `8954`**.**

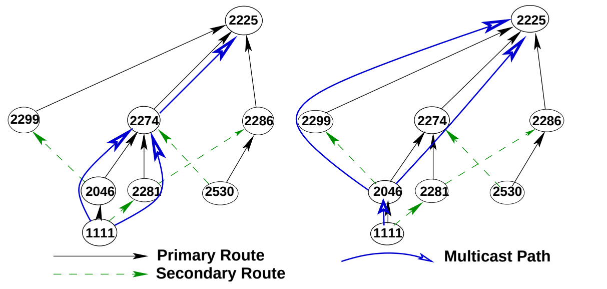

Figure 5. ***Constrained Multicast.*** **Two examples of constrained multicast showing the multicast occurring at different positions on the overlay path.**

tured P2P protocol (Tapestry), constructed with and without overlay hop latency information.

Link Quality Estimation: To measure the quality of routing links, nodes send periodic beacons on outgoing links. At longer periodic intervals, each node replies with an aggregated acknowledgment message. Each ack includes sequence numbers for beacons received, allowing the sender to gauge overall link quality and loss rates. Backup routes need to be probed periodically as well. To conserve bandwidth, we send beacons to primary entries using one beacon rate, and probe backup entries at half that rate.

We derive an estimated link quality from the current measured loss rate and a history of past values. To avoid overreacting to intermittent problems (and avoid route flapping), we introduce damping by estimating loss rate as:

$$L_n = (1-\alpha)\cdot L_{n-1} + \alpha \cdot L_p \qquad (2)$$

where L_p is an instantaneous loss rate from the current period, and α is the hysteresis factor. We explore the appropriate damping factor in Section 4.

2.3. Resilient Routing Policies

Having described mechanisms to maintain and monitor backup paths, we need to define how such paths are used to evade routing failures. We describe here two policies that

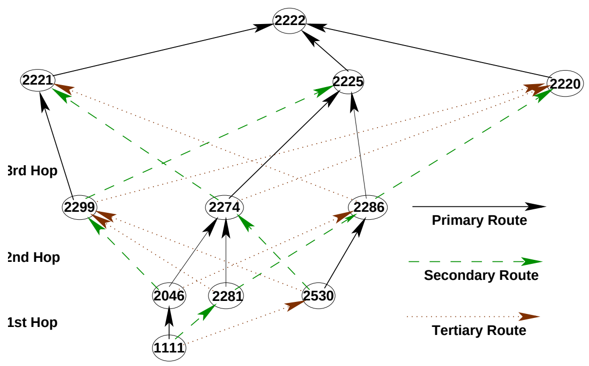

Figure 6. ***Path convergence with prefix routing.*** **Routing path from** `5230` **to** `8954` **in prefix-based protocol. Note that with each additional hop, the expected number of nearby next hop routers decreases, causing paths to rapidly converge.**

define how routes are chosen under failure and lossy conditions. In our discussions, we refer to *primary* and *backup* entries in the routing table, where backups are next hop nodes that satisfy the routing constraint but are further away in the network.

First Reachable Link Selection: We first define a simple policy called *First Reachable Link Selection* (FRLS), to route messages around failures. A node observes link or node failures as near-total loss of connectivity on an outgoing route. From a set of latency sorted backup paths, FRLS chooses the first route whose link quality is above a defined threshold T_{frls}. See Figure 4 for two examples.

Constrained Multicast: Simple link selection is less effective when multiple links are experiencing high loss. We propose *constrained multicast*, where a message entering a lossy region of the network is duplicated, and the copies are sent on multiple outgoing hops. Constrained multicast is complementary to FRLS, and is triggered when no next hop path has estimated link quality higher than T_{frls}.

For example, a node monitors three possible paths to the next hop, (A, B, and C), and stores and sorts them sorted by latency. A typical T_{frls} might be 70%. After a link failure on A, estimated link qualities might be 5% (A), 95% (B) and 85% (C). FRLS chooses the first link in order with the minimum link quality (B). In case of high loss, link qualities might be 45%, 40%, and 60%. Since no path satisfies T_{frls}, messages are duplicated and sent on some subset of the available paths.

Figure 5 shows two examples of constrained multicast occurring at different points in the routing path. A discussion of routing policies that provide a controlled tradeoff between bandwidth and reliability is available in [31].

While naive use of constrained multicast can exacerbate lossy links when loss is due to congestion, the additional traffic is sent on an alternate (likely independent) path. Additionally, we now discuss a convergence property that can significantly reduce the bandwidth overhead imposed by duplicate messages, by allowing us to selectively detect and drop duplicates.

Efficiency via Path Convergence: We observe that the overhead of routing away from the primary path can be controlled on protocols that demonstrate "path convergence," where paths to a common destination intersect at a rate proportional to the distance between the source nodes. Figure 6 shows path convergence in Tapestry.

This property is exhibited in protocols that consider network proximity in conjunction with ID-based constraints on intermediate overlay routers. Pastry and Tapestry are protocols that implement proximity routing based on a prefix routing scheme. With each additional hop, the expected number of nodes in a network that qualify as a next hop router decreases linearly. Therefore, nearby nodes are increasingly likely to choose a common next hop as they zero in on their destination by name and the number of possible routers decreases. Recent work shows that underlying network geometries for most protocols demonstrate path convergence properties [8].

With path convergence, a message that takes a backup route is likely to converge back to the primary path on the next hop. This minimizes the impact of taking a single backup path on end-to-end latency. For constrained multicast, convergence allows routers to detect and drop duplicate packets, minimizing the stress put on the network by the duplicate. Routers identify messages by a flow ID and a monotonically increasing sequence number. Each router can effectively detect and drop duplicates with efficient use of a finite queue of sequence numbers.

Self-Repair: While much of our discussion has focused on routing on alternate paths when network links have failed, we note that self-repair algorithms must be present to replenish backup paths after recovering from a failure. Otherwise, primary and backup paths will all eventually fail, leaving some paths unreachable. When the overlay detects any path failure, it must act to replace the failed route and restore the pre-failure level of path redundancy.

Algorithms for self-repair are specific to each overlay protocol. In general, their goal is to find additional nodes with a specific constraint (matching a certain prefix or having a certain position in a linear or coordinate namespace), given some nodes with that property. Two general strategies are possible. A node can query nearby nodes to minimize repair latency, or query other nodes that already satisfy the constraint. The latter strategy gives a much higher chance of a successful repair, at the cost of contacting further away nodes. For example, a Tapestry node can query nearby nodes for nodes that match prefix P, or query nodes in its routing table that already share P for similar nodes.

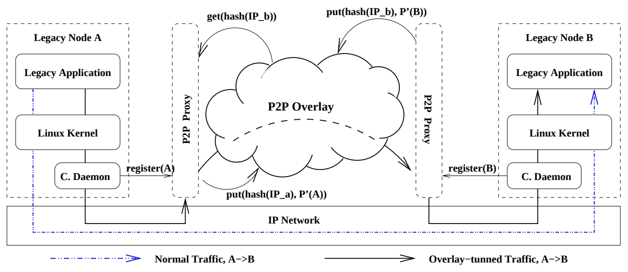

Figure 7. ***Proxy architecture.*** **Architectural components involved in routing messages from source A to destination B. Destination B stores its proxy ID with a hash of its IP address as an object in the overlay. The source proxy retrieves B's proxy ID from the overlay and routes A's traffic to it.**

3. Tunneling Traffic via Structured Overlays

In this section, we discuss an architecture that tunnels legacy application traffic through the P2P overlay so that it can benefit from the fault tolerance of Section 2; tunneled traffic leverages overlay mechanisms to route around link and node failures.

3.1. Transparent Tunneling

Figure 7 shows an architecture that tunnels application traffic through an overlay via nearby proxy nodes. Clients contain overlay-aware daemons that make themselves addressable in the overlay by locating nearby proxies and advertising mappings between their native IP addresses and overlay proxy addresses. With each new outgoing IP connection, a client daemon determines whether the destination is reachable through the overlay; if so, the daemon redirects traffic to the nearby proxy where it enters the overlay, routes to the destination proxy, then exits to the destination node. We elaborate in the following paragraphs.

Proxy traffic redirection: Traffic redirection involves two steps, registering a routeable ID for each legacy node in the overlay ID space, and publishing a mapping from the node's IP address to that ID. To register an ID, the daemon on the legacy node (A) first chooses a nearby overlay node as its proxy using an introduction service or out-of-band directory. Recall that in a structured P2P overlay, IDs in the namespace are mapped to a specific "root" node. The proxy (P) assigns A an ID in the ID space: ($P(A)$), such that $P(A)$ is the closest unused id to P inside its range, where range is defined by the routing protocol. Figure 8 illustrates registration and tunneling.

For example, legacy nodes registering with a Chord proxy would receive sequentially decreasing identifiers beginning with $P - 1$. This insures that messages addressed to $P(A)$ are delivered to P despite changes in the overlay membership. Assuming nodeIDs are assigned uniformly at random, the probability that a given proxy node with l

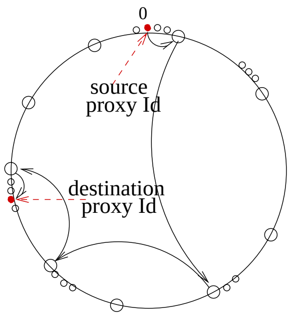

Figure 8. ***Registering with proxy nodes.*** **Legacy application nodes register with nearby proxies and are allocated proxy IDs which are *close* to the name of the proxy node. Legacy nodes can address each other with these new proxy names, routing through the overlay to reach one another.**

legacy clients loses one of them to a new overlay node is l/N where N is the size of the namespace. Given p active proxy nodes, the chance of any proxy losing a legacy node is then $(l \cdot p)/N$. For example, in a 160 bit namespace overlay of 10,000 nodes each averaging 10 legacy clients, the probability of a new node "hijacking" one of them from its proxy is $(10 \cdot 10,000)/2^{160}$, or 2^{-143}.

The next step is to establish a mapping from A's IP address to its overlay ID. This allows the overlay to do a "DNS-like" name translation. Since most structured P2P overlays already have a storage layer such as a Distributed Hash Table (DHT), we opt to utilize that instead. Therefore, the proxy stores in the overlay a mapping between a hash of the legacy node's IP and its proxy identifier ($<$SHA-1(IP $_A$), $P(A)$ $>$), either using the *put* call on a DHT interface, or by storing the mapping locally and using the *publish* call on the DOLR interface [6].

Application Interface at Endpoints: The client-side daemon is implemented as a set of packet forwarding rules and a packet encapsulation process. The daemon can capture packets using general rules in Linux *IP-chains* or FreeBSD *divert sockets*. It processes them, using IP-IP encapsulation to forward certain packets to the proxy and the remainder unchanged back onto the normal network interface.

The daemon has two responsibilities: register the local IP address as a routeable destination in the overlay and divert appropriate outgoing traffic to the overlay. We expect IP registration to occur when the daemon starts, as described above. Since not all destinations will have made themselves reachable through the overlay, the daemon monitors outgoing traffic and selects flows for tunneling. This can be done in a local, user-transparent fashion by hijacking all DNS requests and new connection requests. We query the new IP addresses to determine whether they are reachable via the overlay, and cache the result. When the application starts a connection to an address routeable by the overlay, the daemon notifies the proxy, which locates the destination node's proxy identifier by performing a *get (SHA-1(IP $_{dest}$))*.

Redundant Proxy Management: While the overlay provides a scalable way to route around failures in the network, a proxy may still fail or become disconnected from the overlay or the destination nodes it is responsible for. We outline three possible solutions. In each case, a legacy node L registers with a small number (n) of proxy nodes, sorted in order by a policy-driven metric such as latency to L, available bandwidth, or proxy load (defined by node registrations or bandwidth consumption). The overlay maps the destination IP to its set of destination proxy identifiers. The sender proxy also caches this information during connection setup.

The naive solution assumes that the overlay returns an error for each undeliverable message, which are resent from the sender-side proxy to the next entry on the destination's identifier list. This solution requires buffering at the sender's proxy, and incurs a roundtrip delay after failure. An alternate solution embeds the backup proxy identifiers inside each message. As a message encounters a failed hop, it replaces its destination with the next identifier from the list, and tries to route to that proxy. The additional routing logic can be implemented on top of the proposed common up-call interface for structured P2P overlays [6].

An alternate solution is to use RAID-like striping to send the data stream across multiple proxies. The source proxy can send each of $n-1$ sequential packets to proxies on the identifier list, and a bit-wise XOR parity block to the n^{th} proxy. Any missing packet can be reconstructed from the remaining packets. This design provides fast and transparent recovery from single proxy failures at the cost of an additional $1/(n-1)$ proportional bandwidth.

3.2. Challenges to Deployment

Finally, we consider issues that arise when deploying fault-resilient overlays across the Internet. Since overlay nodes function as application-level traffic routers, they require low-latency, high-bandwidth connectivity to the network. We expect Internet Service Providers (ISPs) to deploy these overlays on their internal networks and offer resilient traffic tunneling as a value-added service to their customers. An ISP chooses the number and location of traffic overlay nodes. Adding overlay nodes in the interior increases the number of backup paths and overlay resiliency, while placing nodes closer to customers reduces the likelihood of failures between the client and the overlay.

One issue is that a deployed overlay is limited by the reach of the ISP's network. Connections that cross ISP boundaries require a cross-domain solution. One possibility is for smaller ISPs to "merge" their overlays with those of larger ISPs, allowing them to fully share the namespace and share routing traffic. A 160-bit namespace ensures that the probability of namespace collisions will remain statistically

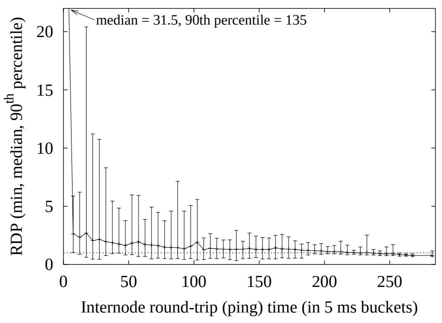

Figure 9. *Tapestry Performance.* Relative Delay Penalty (RDP) for packets routing the Tapestry network on the PlanetLab network testbed. Tapestry provides relatively low overhead compared to IP in the wide area.

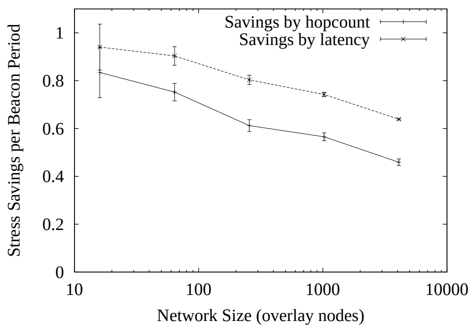

Figure 10. *Maintenance Advantage of Proximity (Simulation).* Proximity reduces relative bandwidth consumption (TBC) of beacons over randomized, prefix-based routing schemes.

insignificant. A second solution is to set up well defined peering points between each ISP's overlay by using wide-area routing similar to that proposed by the Brocade interdomain overlay work [30]. Peering points can form their own overlay and advertise local addresses as objects on the secondary overlay. The resulting hierarchy has properties similar to BGP. Further comparisons are beyond the scope of this paper.

4. Evaluation of Adaptive Mechanisms

To explore the potential for adaptive fault tolerance as described in previous sections, we present simulation results as well as measurements from a functioning Tapestry system [32]. Tapestry comprises 55,000 lines of Java written in event-driven style on top of SEDA [28] for fast, non-blocking I/O. This version of Tapestry[1] includes all of the mechanisms of Section 2, including components for beacon-based fault detection across primary and backup paths, first-reachable link selection (FRLS), and constrained multicast with duplicate packet detection.

In this section, we take an in-depth look at the impact of the adaptive mechanisms proposed in this paper. Starting with baseline Tapestry, we examine the impact of the proposed routing policies through both simulation and experimental measurements. We also test Tapestry's ability to exploit underlying network redundancy, and explore the tradeoff between beacon rate and responsiveness to failures.

Tapestry has been deployed on the PlanetLab testbed [20], a network of 160 machines at 65 sites worldwide. To gain additional nodes, we invoke multiple virtual nodes per physical node. Figure 9 illustrates the basic routing performance of the Tapestry implementation by measuring the increase in latency that a packet experiences when tunneling through the overlay. The latency overhead is low, particularly for wide-area routes. The high variability on short routes is due to a combination of virtual nodes competing for highly loaded CPUs and queuing delays in the event handling layer [32].

4.1. Analysis and Simulation

To evaluate the potential for adaptation, we start by examining several microbenchmarks in simulation. To do this, we implemented a network simulator using the Stanford Graph Base libraries [14]. In the following measurements, we utilize seven different 5000-node transit stub topologies; unless otherwise specified, we then construct Tapestry overlay networks of size 4096 nodes against which to measure our results.

Proximity Routing and TBC: As mentioned previously, Tapestry provides proximity routing. We start by illustrating the advantage of proximity-based structures in reducing the Total Bandwidth Consumption (TBC) of monitoring beacons[2]. To perform the comparison, we construct overlays of different sizes both with and without proximity; without proximity means that we construct random topologies that adhere to the basic prefix-routing scheme but which do not utilize network proximity in their construction. The TBC saving with a proximity-enabled overlay is plotted in Figure 10. We see that maintenance traffic with proximity routing provides a significant reduction (up to 50%) in resources. Section 4.2 will explore the absolute amount of maintenance traffic.

Overhead of FRLS: To ensure our resilient routing policies do not impose unreasonable overhead on tunneled traffic, we simulate their impact on end-to-end latency and bandwidth consumption through simulation. We first measure

1 The Tapestry implementation is available for public download at `http://oceanstore.cs.berkeley.edu/`.

2 Recall from Section 2.2 that the TBC is computed by multiplying the beacon bit rate by distance—either in number of IP hops or latency.

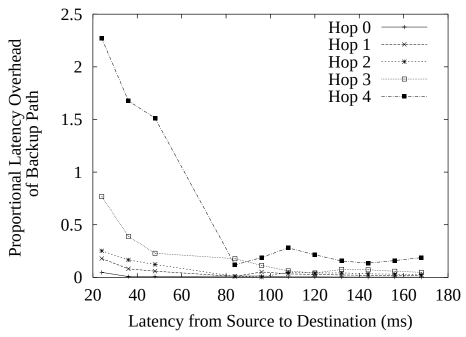

Figure 11. ***Latency Cost of Backup Paths (Simulation).*** **Here we show the end-to-end proportional increase in routing latency when Tapestry routes around a single failure.**

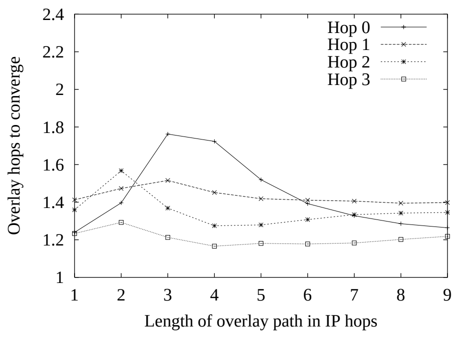

Figure 12. ***Convergence Rate (Simulation).*** **The number of overlay hops taken for duplicated messages in constrained multicast to converge, as a function of path length.**

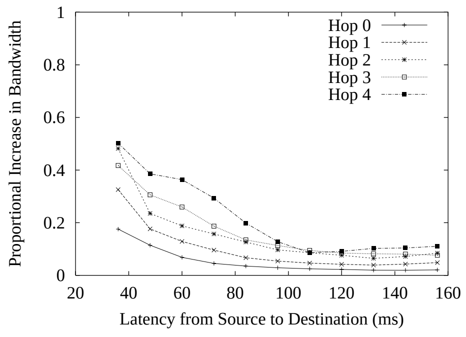

Figure 13. ***Bandwidth Overhead of Constrained Multicast (Simulation).*** **The proportional increase in bandwidth consumed by using a single constrained multicast.**

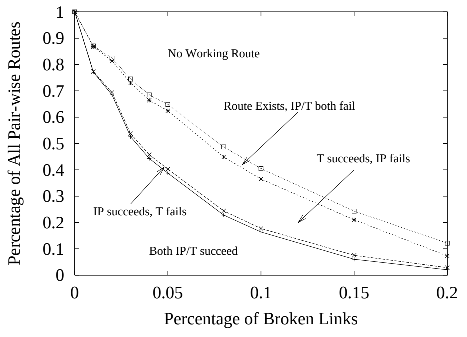

Figure 14. ***Routing Around Failures with FRLS.*** **Simulation of the routing behavior of a Tapestry overlay (2 backup routes) and normal IP on a transit stub network (4096 overlay nodes on 5000 nodes) against randomly placed link failures.**

the increase in latency we incur by using FRLS to route around a failure. We expect that by taking locally suboptimal backup routes, we are increasing end-to-end routing latency. Figure 11 shows the proportional increase in routing latency when routing around a single failure. We see that when backup routes are taken closer to the destination (3^{rd} or 4^{th} on a 6 hop overlay path), the overhead incurred is higher. Overall, the latency cost is generally very small ($<$ 20% of the end-to-end path latency).

Constrained Multicast and Path Convergence: We continue by quantifying the expected bandwidth cost of constrained multicast, assuming a protocol with path convergence. First, we verify that Tapestry routing provides path convergence. Path convergence allows us to limit the amount of bandwidth consumed by duplicate messages in constrained multicast. As Figure 12 shows, duplicate messages generally converge with the original after 1 hop, minimizing additional bandwidth used.

Next, Figure 13 measures the additional bandwidth consumed by the duplicated packets, assuming they are dropped when they converge paths with the originals. The additional bandwidth is plotted as a ratio to the end-to-end bandwidth consumed. Again, failures closer to the destination are more costly, but the bandwidth overhead is generally low ($<$ 20%).

Reachability Simulation: In this experiment, we simulate the impact of random link failures on a structured overlay in the wide area. We construct a 4096 node Tapestry topology on a 5000-node transit stub network, using base 4 digits for prefix routing and 2 backups per routing entry.

We monitor the connectivity of pair-wise paths between all overlay nodes, and incrementally inject randomly placed link errors into the network. After each new set of failures is introduced, we measure the resulting connectivity of all paths routed through IP (estimated by the shortest path) and through Tapestry with FRLS. We assume a time frame of one or two seconds, tolerable delay for buffered multimedia applications, but insufficient time for IP level route convergence. We plot the results in Figure 14 as a probability graph, showing the proportion of all paths which succeed or fail under each protocol. The results show that Tapestry routing performs almost ideally, succeeding for the large majority of paths where connectivity is maintained after failures. Cases where Tapestry routing fails to find an existing path are rare.

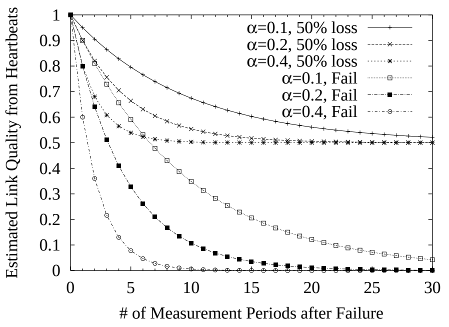

Figure 15. *Hysteresis Tradeoff.* A simulation of the adaptivity of a function to incorporate hysteresis in fault estimation using periodic beacons. Curves show response time after both a link failure and a loss event causing 50% loss.

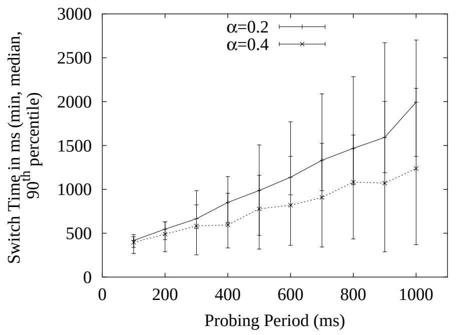

Figure 16. *Route Switch Time vs. Probing Frequency.* Measured time between failure and recovery is plotted against the probing frequency. For this experiment, the hysteresis factors $\alpha = 0.2$ and $\alpha = 0.4$ are shown.

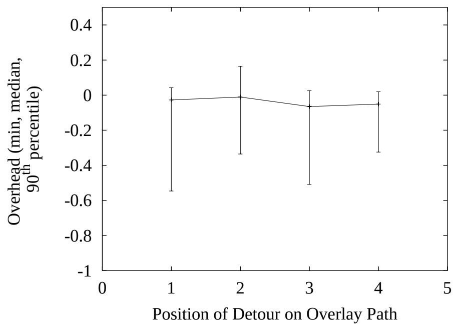

Figure 17. *Overhead of Fault-Tolerant Routing.* The increase in latency incurred when a packet takes a backup path. Data separated by which overlay hop encounter the detour. Pairwise overlay paths are taken from PlanetLab nodes, and have a maximum hop count of six.

4.2. Microbenchmarks of a Deployed System

Next, we use microbenchmarks to illustrate properties of the Tapestry implementation deployed on PlanetLab. To probe Tapestry's adaptation behavior, we implemented a fault-injection layer that allows a centralized controller to inject network faults into running nodes. Nodes can be instructed to drop some or all incoming network traffic based on message type (*e.g.* data or control) and message source; we then report the results. In general, we present median values, with error bars representing 90^{th} percentile and minimum values. We use the 90^{th} percentile values to remove outlier factors such as garbage collection and virtualization scheduling problems.

Failover Time: Our first microbenchmark measures the correlation between fail-over time and length of the beacon period. We start by selecting an appropriate hysteresis factor α for link quality estimation (Equation 2). Figure 15 illustrates how quickly estimated values converge to actual link quality for different values of α and link loss rate. Using this, we can see that an α value between 0.2 and 0.4 provides a reasonable compromise between response rate and noise tolerance.

For the experiment, we deploy a small overlay of 4 nodes, including nodes at U.C. Berkeley, U. Washington, U.C. San Diego and MIT. NodeIDs are assigned such that traffic from Berkeley routes to MIT via Washington, with UCSD as backup. The round-trip distance of the failing link (Berkeley-Washington) is approximately 30ms.

Using hysteresis factors of 0.2 and 0.4, we inject faults at random intervals, and measure the elapsed time to detect and redirect traffic around the fault. Figure 16 plots the min, median and 90^{th} percentile values against the beacon period. As expected, switch-over time scales linearly to the probing period with a small constant. With a reasonable beaconing period of 300ms, response times for both α values (660ms and 580ms) are well within the acceptable limits of interactive applications, including streaming multimedia. No messages were lost after traffic was redirected.

Redirection Penalty: To quantify the latency cost in redirecting traffic onto a backup path, we deploy a Tapestry network of 200 nodes using digits of base 4 across the PlanetLab network. We probe all pair-wise paths to select a random sample of source-destination pairs with sufficiently distinct IDs to require five overlay hops. On each path, we measure the change in latency resulting from taking a single backup path, plotted against the hop where the backup path was taken. The results are shown in Figure 17.

The results mirror results shown in Figure 11, and confirm that taking a single backup path has little impact on end-to-end routing latency. In fact, because of the small number of nodes in our PlanetLab overlay, taking an alternate path can sometimes shorten the number of hops and reduce overall latency. For example, given 3 nodes at Duke (000), Georgia Tech (213) and MIT (222), a route from Duke to MIT would point to Georgia Tech as the optimal first hop and keep MIT as a backup. If a failure occurs on

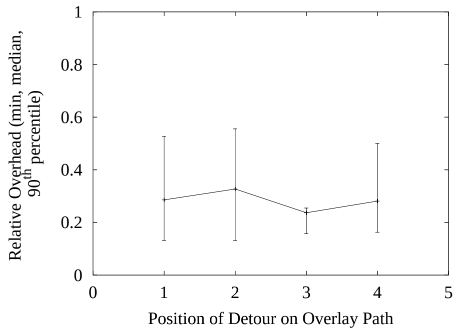

Figure 18. ***Overhead of Constrained Multicast.*** **The total bandwidth penalty for sending a duplicate message when loss is detected at the next hop, plotted as a fractional increase over normal routing. Data separated by which overlay hop encounters the split.**

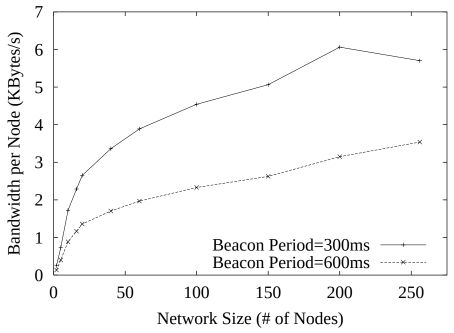

Figure 19. ***Cost of Monitoring.*** **Here we show bandwidth used for fault-detection as a function of overlay network size. Individual curves represent different monitoring periods, and bandwidth is measured in kilobytes per second per node.**

the primary route, a message will use the backup path and route directly to MIT, improving end-to-end latency. This explains the low minimum values in Figure 17.

Constrained Multicast Penalty: To characterize the cost of constrained multicast, we start with a deployed network of 200 nodes on PlanetLab. We select a group of paths with 5 overlay hops, and plot the bandwidth overhead as a function of the hop where the duplicate message was sent out. Without knowledge of IP level routers under PlanetLab, we approximate the TBC metric by using the bandwidth latency product. Figure 18 shows that our deployed prototype performs as expected, with duplicate messages incurring less than 30% of the end-to-end bandwidth consumption. This figure reports the proportional increase in total bandwidth consumption (TBC) over the original path and can be compared with simulation results in Figure 13.

Beaconing Overhead: We quantify the periodic beaconing overhead of our Tapestry implementation by measuring the total bandwidth used by beacon messages, and plotting that against the size of the overlay network. Figure 19 shows the result as kilobytes per second sent by each node in the overlay, using a beacon period of 300ms. Each routing entry has two backups, each beaconed every 600ms. The bandwidth used is low and scales logarithmically with the network size. Furthermore, our measurements are consistent with bandwidth estimates in Section 2.2. Note that along with Figure 16, this figure shows that moderate to large overlays can respond to link failures in under 700ms, while keeping beaconing traffic low ($<$7KB/s).

4.3. The Importance of Self-Repair

While low-rate transient failures can be handled with techniques from Section 2, long term stability depends crucially on mechanisms that *repair* redundancy and *restore* locality. Without continuous “precomputation” and path discovery, path redundancy will slowly degrade as backup routes fail over time. In the case of Tapestry, this means that entries in the routing table must be refreshed at a rate that keeps ahead of network failures and changes.

To illustrate this point, we deploy Tapestry nodes in a LAN cluster and subject them to abrupt and continuous change; since this experiment focuses on fault resilience and not latency, we do not impose a network topology or packet delay on the system. We then measure overlay *connectivity* by measuring the rate of success in routing requests between random pairs of IDs in the namespace. The result is shown in Figures 20 and 21.

Both of these figures illustrate an experiment that initializes the network with 150 nodes, then introduces a massive failure event at T=5 minutes by manually killing (kill -9) 30 nodes. At T=10 minutes, we add 75 nodes to the network in parallel. Finally, at T=15 minutes, all nodes in the overlay begin to participate in a random churn test, where every 10 seconds nodes enter and leave the network according to a randomized stochastic process, each with a mean duration of 2 minutes in the network.

We plot the number of nodes in the system along with average query latency and the success rate of routing requests. Without repair, Figure 20 shows that routing success rate quickly degrades after the massive fail event, and never recovers. Furthermore, nodes in the churn test slowly lose their redundant paths as neighbors leave the network, leading to a steady decline in route connectivity. In contrast, Figure 21 shows that Tapestry with self-repair quickly recovers after massive failure and join events to restore routing success to 100%. Even under constant churn, our algorithms repair routes fast enough to maintain a high level of routing availability. Note that the relatively low routing latency shown in Figure 20 is due to the fact that inconsistent routes lead to a portion of the massive join failing, result-

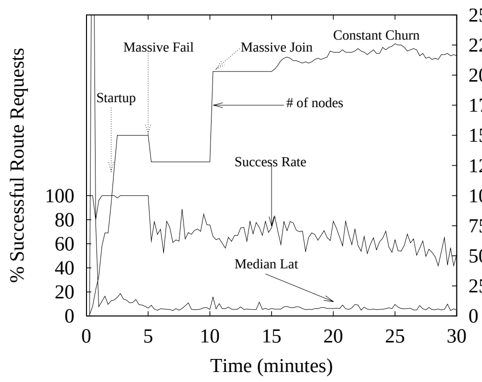

Figure 20. *Pair-wise Routing without Repair.* Success rate of Tapestry routing between random pairs of nodes with self-repair mechanisms disabled during massive failure, massive join, and constant churn conditions.

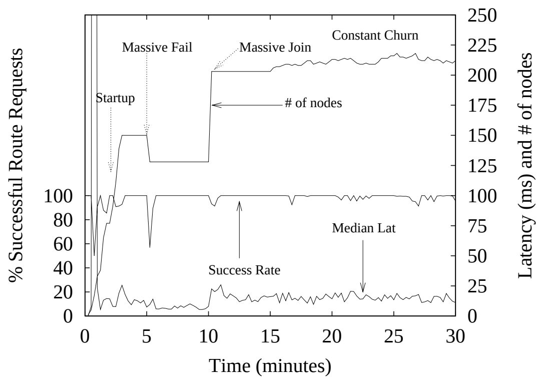

Figure 21. *Pair-wise Routing with Self-Repair.* Success rate of Tapestry routing between random pairs of nodes with self-repair enabled during massive failure, massive join, and constant churn conditions.

ing in shorter routes and lower latency for requests that succeed.

4.4. Putting It All Together

Experiments and analysis quantify the benefits of our proposal, and verify its feasibility. While simulations show that FRLS exploits the majority of routing redundancy present in a network, measurements show that our prototype can adapt to failure in under 700 milliseconds, even with a reasonable dampening α factor. Furthermore, the bandwidth required to support such high adaptivity is low. Finally, we show that in addition to circumventing link failures in the network, the overlay self-repairs following node failures in order to maintain high availability.

5. Related Work

Structured peer-to-peer overlays provide scalable, load-balanced routing of messages to objects or endpoints. The design of the first protocols ([21, 25, 27, 32]) have led to the design of scalable wide-area applications ([4, 5, 24]) and the development of new protocols with specialized properties [9, 12, 18, 29]. Our work requires only the key-based routing API [6], and can be implemented on a number of these protocols.

Resilient Overlay Networks (RON) allow application traffic tunneling through a small number of overlay nodes and demonstrates its feasibility [1]. Its pair-wise communication results in $O(N^2)$ messages and limits its scale to tens of nodes. The Internet Indirection Infrastructure (I^3) uses triggers embedded in the infrastructure to redirect application level traffic for mobility and resilience. Unlike redirection points in our work, I^3's trigger placement is done manually. The Secure Overlay Services (SOS) [13] project places protected servers inside the overlay domain, and filters tunneled application traffic at the edge nodes to prevent denial of service attacks. The Detour project proposes a network of deployed routers that direct tunneled traffic to study efficiency and loss rates of Internet routing paths [26]. Detour focuses on using routers to gather network information to benefit the transport protocol, and does not provide scalable methods for detecting faults and managing backup paths.

Previous research quantified the loss of network connectivity on the wide-area network. Bharat *et. al.* performed a quantitative analysis of service availability across a WAN and developed a failure model parameterized by failure location and failure duration [3]. This work focused on understanding the cause and consequences of BGP failures. Labovitz *et. al.* examined the latencies in Internet path failure, fail-over, and repair resulting from the convergence properties of BGP routing algorithms [15]. Recent work by Mahajan *et. al.* quantified the occurence of BGP misconfigurations by measurement and by ISP surveys [17].

6. Conclusion

In this paper, we show that structured peer-to-peer (P2P) overlay networks can provide efficient, responsive fault resilient routing to legacy applications. These overlays enable precomputation of backup paths and efficient use of soft-state beacons for fault detection. While we illustrate our ideas using a prototype of the Tapestry overlay network, our designs are directly applicable to other P2P protocols.

At each routing hop, Tapestry routing chooses between optimal or near optimal local paths, while soft-state beacons continuously probe the network to detect failures and maintain path redundancy. We showed that two simple routing policies can be used to achieve near-optimal fault-resilience while incurring low overhead in terms of latency and bandwidth relative to a fault-free network. A moderate sized overlay can respond to link failures in under 700 milliseconds while using less than 7 Kilobytes/second of beaconing traffic, agile enough to support most streaming multimedia applications. These techniques show great promise for addressing the reliability and responsiveness needed by today's global-scale network applications.

7. Acknowledgements

We would like to thank Chen-nee Chuah for her helpful insights on IS-IS. We also want to thank the anonymous reviewers and Quyen Ta for their feedback and suggestions.

References

[1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. of SOSP*. ACM, Oct 2001.

[2] T. Bu, L. Gao, and D. Towsley. On routing table growth. In *Proc. of Global Internet Symposium*. IEEE, 2002.

[3] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-end WAN service availability. In *Proc. of USITS*. USENIX, Mar 2001.

[4] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. In *Proc. of OSDI*. ACM, Dec 2002.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of SOSP*. ACM, Oct 2001.

[6] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured P2P overlays. In *Proc. of IPTPS*, Berkeley, CA, Feb 2003.

[7] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek. Measuring the effects of internet path faults on reactive routing. In *Proc. of SIGMETRICS*. ACM, June 2003.

[8] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Schenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. of SIGCOMM*, Karlsruhe, Germany, Sep 2003. ACM.

[9] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. of USITS*, Seattle, WA, Mar 2003. USENIX.

[10] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proc. of SPAA*, Winnipeg, Canada, Aug 2002. ACM.

[11] G. Iannaccone, C.-N. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an IP backbone. In *Proc. of the Internet Measurement Workshop*, Marseille, France, Nov 2002. ACM.

[12] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *Proc. of IPTPS*, Berkeley, CA, Feb 2003.

[13] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *Proc. of SIGCOMM*, Pittsburgh, PA, Aug 2002. ACM.

[14] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. ACM Press and Addison-Wesley, New York, 1993.

[15] C. Labovitz, A. Ahuja, A. Abose, and F. Jahanian. Delayed internet routing convergence. In *Proc. of SIGCOMM*. ACM, Aug 2000.

[16] C. Labovitz, A. Ahuja, R. Wattenhofer, and S. Venkatachary. The impact of internet policy and topology on delayed routing convergence. In *Proc. of INFOCOM*. IEEE, 2001.

[17] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *Proc. of SIGCOMM*, Pittsburgh, PA, Aug 2002. ACM.

[18] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. of PODC*. ACM, 2002.

[19] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of IPTPS*, Mar 2002.

[20] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proc. of HotNets-I*. ACM, 2002.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. of SIGCOMM*. ACM, Aug 2001.

[22] Y. Rekhter and T. Li. *An Architecture for IP Address Allocation with CIDR*. IETF, 1993. RFC 1518, `http://www.isi.edu/in-notes/rfc1518.txt`.

[23] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). *IEEE Micro*, 19(1):50–59, Jan. 1999. Also IETF RFC 1771.

[24] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The OceanStore prototype. In *Proc. of FAST*, San Francisco, Apr 2003. USENIX.

[25] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*. ACM, Nov 2001.

[26] S. Savage et al. Detour, a case for informed internet routing and transport. *IEEE Micro*, 19(1):50–59, Jan 1999.

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM*. ACM, Aug 2001.

[28] M. Welsh, D. Culler, and E. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Proc. of SOSP*, Banff, Canada, Oct 2001. ACM.

[29] U. Wieder and M. Naor. A simple fault tolerant distributed hash table. In *Proc. of IPTPS*, Berkeley, CA, Feb 2003.

[30] B. Y. Zhao, Y. Duan, L. Huang, A. Joseph, and J. Kubiatowicz. Brocade: Landmark routing on overlay networks. In *Proc. of IPTPS*, Mar 2002.

[31] B. Y. Zhao, L. Huang, J. D. Kubiatowicz, and A. D. Joseph. Exploiting routing redundancy using a wide-area overlay. Technical Report CSD-02-1215, U. C. Berkeley, Nov 2002.

[32] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE J-SAC*, 2003. Special Issue on Service Overlay Networks, to appear.