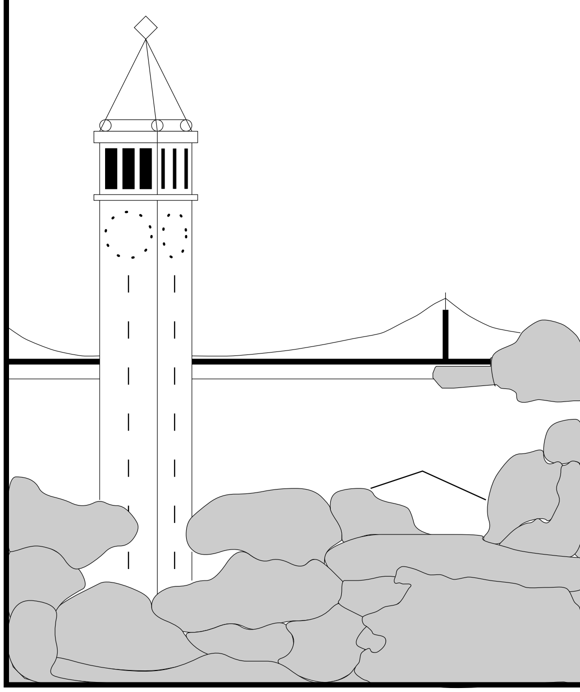
Exploiting Routing Redundancy Using a Wide-area Overlay

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



Report No. UCB/CSD-02-1215

November 2002

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Exploiting Routing Redundancy Using a Wide-area Overlay

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

November 2002

Abstract

As new and interesting peer-to-peer applications combine with advancements in networking technology, they are reaching millions of users across the globe. Numerous studies have shown, however, that loss of connectivity is common on the wide-area network, due to hardware and software failures, and network misconfigurations. Despite the natural redundancy present in underlying network links, the current IP layer fails to recognize and recover from these frequent failures in a timely fashion. This paper presents fault-tolerant routing on the Tapestry overlay network, which exploits existing network redundancy by dynamically switching traffic onto precomputed alternate routes. Furthermore, messages in our system can be duplicated and multicast "around" network congestion and failure hotspots with rapid reconvergence to drop duplicates. Our simulations show fault-tolerant Tapestry to be highly effective at circumventing link and node failures, with reasonable cost in terms of additional routing latency and bandwidth cost.

1 Introduction

The Internet continues to grow at an impressive rate. With each passing day, users deploy new and more interesting wide-area applications, such as peer-to-peer (P2P) file sharing, instant messaging, real-time information collection and distillation, and multimedia applications. The communication patterns of these new applications vary from simple, point-to-point messaging to complex, multi-party, multicast content distribution. These newer applications tend to place heavy demands on the Internet infrastructure – requiring fault tolerance and quick adaptation while simultaneously demanding low latency and high bandwidth. Given that these are base requirements for new applications, we believe that both high performance and fault tolerance should be available to all users.

Unfortunately, it is becoming increasingly difficult to meet these criteria. The sheer size and complexity of the network leads to frequent periods of wide-area disconnection or poor performance. Misconfigurations and hardware faults contribute to these problems. Individual routers vary widely in performance and connectivity, leading to a difficult optimization problem under ideal circumstances. Worse, the network spans many independent administrative boundaries, making the goal of coordinated problem detection and correction an elusive goal at best. While recent network service provider consolidations are reducing ownership/management-related problems, the reliability benefit may be becoming outweighed by the corresponding reduction in path diversity[1].

[1] Consider, for example the chaos caused by the Baltimore tunnel fire in July of 2001.

Further, the probability of *flash crowds*, such as those that result from sudden popularity of a web site or service, increases with the scale of the network. Flash crowds result in localized congestive flows at senders or receivers and cause periods of high delay or loss. Most automated solutions to flash crowds are expensive and available to large enterprises rather than individuals. Automated approaches to handling network hot spots are either based on CDN approaches (useful for one-way data distribution), proprietary networks (*e.g.*, Metricom [3]), single-user solutions (*e.g.*, RON [1]), or are mostly ad hoc and similarly limited in reaction time (*e.g.*, BGP-based solutions [15]).

We expect that as the network grows in scale and scope, wide-area disconnections and poor performance will become more common. Today, many faults and performance problems are dealt with manually, a solution that imposes a significant time delay, does not scale well with the size of the Internet, and is not available to individual users. Even traditional automated approaches to detecting and routing around faults (*e.g.*, the Border Gateway Protocol [15]) may take up to 30 minutes to react to and isolate a fault.

Previous work in [2] demonstrated the need for high availability for network services, and outlines their belief in the generalized approaches of dynamic service replication and migration, and dynamic routing around network hot spots and faults. One of the most significant delays in BGP adaptation results from the time that it takes to recalculate routes and pair-wise disseminate this information. Precomputing alternate pathways is an obvious solution – one that was taken by RON [1] – but must be done in a scalable fashion, maintaining as much communication locality as possible.

In this paper, we seek to provide fault tolerance and high performance through a two-pronged approach: (1) continuous precomputation of alternative pathways and (2) dynamic selection among alternates. We start with a routing scheme that is amenable to alternate path computation and selection: the Tapestry Distributed Object Location and Routing (DOLR) service [5, 24]. This overlay framework incorporates multiple simultaneous paths between any two nodes in the network (*i.e.*, Tapestry selects from several optimal or near optimal paths at each routing hop). As a result, we can decouple the discovery of backup paths ("pre-computation") from rapid adaptation in response to failure or congestion. Tapestry continuously monitors the connections and performance between routing peers using a soft-state, heartbeat-based approach and dynamically pre-computes alternate routes. When a fault is detected, the network either switches to an alternate route or multicasts traffic across two or more routes.

The design of the Tapestry routing algorithm ensures that "mis-routed" traffic rapidly converges back onto the optimal path to the destination with minimal excess traffic, even when traffic is multicast across multiple links. We show via simulation that a simple protocol can be used to achieve near-optimal fault-resilience, a significant improvement over IP routing. Furthermore, routing around failures incurs low overhead in terms of latency and bandwidth relative to the original path. We also show that "mis-routed" traffic converges quickly, implying that multicasting traffic across multiple routes is feasible and provides increased reliability at a relatively low bandwidth cost.

In the rest of the paper, we examine related work in availability, fault-resilient network routing, and P2P distributed location and routing services in Section 2. Then we present the basic Tapestry routing algorithm with its redundancy primitives in Section 3, followed by in-depth details of the proposed fault-resilient routing mechanisms in Section 4. Finally, we show simulation results in Section 5, discuss additional issues in Section 6, and conclude in Section 7.

2 Related Work

The work presented in this paper is related to several projects in wide-area application availability, wide-area routing failures, fault-tolerant route-around overlays, and decentralized object location systems. In this section, we describe the key related projects (to the best of our knowledge) and provide points of differentiation for fault-tolerant Tapestry.

Bharat *et. al.* performed quantitative analysis of service availability across a wide-area network [2] and developed a failure model that was parameterized by failure location and failure duration. Using trace-based simulation, they proposed and examined several techniques for improving end-to-end service availability by masking network failures, including data caching, prefetching, and using alternate network paths to route around failures. They conclude that only by combining several techniques together will some systems be able to effectively improve availability.

In order to scale to millions networks, routing in the current Internet is organized in a two-level hierarchy: intra-domain routing and inter-domain routing. Inter-domain routing mainly relies on BGP to exchange reachability information and maintain routing tables [15], however for policy reasons, information shared through BGP between different ISPs is heavily filtered and summarized. As a result, many topological details, especially those for redundant links are hidden. Also, because BGP uses route flap damping and an incremental dissemination mechanism, it may take BGP fault recovery several minutes (3 to 30 minutes) before routes converge to a consistent form. Unfortunately, these delays are on a time-scale that clearly exposes applications to router and link faults, and forces the applications to deal with the faults, often using ad hoc and non-scalable mechanisms.

Paxson studied the large-scale behavior of routing in the Internet [10, 11] and found several routing pathologies, including routing loops and instances of infrastructure failures. He found numerous outages of durations of 30 seconds or greater, and that 3.3 percent of all routes had serious problems during 1995. More importantly, he found that the trend was towards worse wide-area behavior.

Labovitz *et. al.* examined the latencies in Internet path failure, fail-over, and repair resulting from the convergence properties of inter-domain Border Gateway Protocol routing algorithms [7, 8]. They conducted a two-year study, injecting 250,000 routing faults at major Internet exchange points and collecting a large amount of routing update information. Their study showed that the Internet inter-domain routing convergence delay is an order of magnitude slower than was previously thought to be the case. Two important observations were: 40 percent of outages took more 30 minutes to repair; and inter-domain routers take tens of minutes to reach a consistent view of the network topology after a fault. During the convergence procedure, Internet applications will loss network connectivity and/or encounter high packet loss and latency. Most significantly, these results show that the Internet does not support effective timely inter-domain fail-over.

There are several related research projects that are exploring mechanisms for fast failure detection, failure route-around, and efficient failure recover, including the Detour and Resilient Overlay Networks projects. The Detour Project at the University of Washington developed the "sting" tool, which uses TCP to determine forward and reverse path packet loss rates [20]. Using this tool, the researchers developed an architecture in which intelligent routers located at key access and interchange points "tunnel" traffic through the Internet. They have shown that the use of these intelligent tunnels can improve performance and availability by aggregating traffic information, shaping bursty traffic flows, and using more efficient routes.

The Resilient Overlay Networks (RON) project has developed an architecture that enables distributed Internet applications to detect and recover from path outages within several seconds [1]. Its key design goal is to allow end-hosts and applications to work cooperatively to gain improved reliability and performance from the Internet. In the RON architecture, a set of application-layer overlay nodes are deployed in different Autonomous System domains (*e.g.*, Sprintlink, AT&T, and Worldcom). The RON nodes monitor the reachability and quality of the Internet paths between themselves, and use this information to decide whether to route packets directly over the Internet or indirectly through other RON nodes, based upon optimization of various application-specific routing metrics. The researchers observed that RON's routing mechanism was able discover alternate paths in the Internet, and to detect, recover, and route around failures in less than twenty seconds on average. These improvements demonstrate the benefits of moving some of the control over routing into the hands of end-systems.

Beside Tapestry, there are several projects working on different approaches to Decentralized Object Location and Routing (DOLR) algorithms, including Kademlia [9], CAN [13], Pastry [17], Chord [21]. All of these architectures use name-based routing to route requests for objects or files to a nearby replica. They all share similarities with Tapestry in providing scalable location services, including the use of soft-state beacons for fault-detection. The focus of these projects, however, is on object location, and not point to point communication. However, as we discuss in Section 6, our fault-resilience mechanisms are general enough to apply to these systems as well.

3 Tapestry Routing Primitives

In this section, we provide a general review of the Tapestry network layer [5, 24], and some of its basic fault-handling mechanisms. Tapestry is one of several recent projects exploring the value of wide-area Decentralized Object Location and Routing (DOLR) services [13, 17, 21]. It enables messages to locate objects and route to them across an arbitrarily-sized network, while using a routing map with size logarithmic to the network namespace at each hop.

As a location service, Tapestry provides network applications with efficient routing of messages to locations of named objects. Such functionality in Tapestry and related projects has given rise to a new class of wide-area applications [4, 6, 18, 19, 25].

The key distinction between Tapestry and other DOLR infrastructures, however, is its support for point-to-point routing between named nodes. Tapestry uses similar mechanisms to the hashed-suffix mesh introduced by Plaxton, Rajaraman and Richa in [12]. Tapestry routes messages between named nodes across an arbitrarily-sized network using a routing map with size logarithmic to the network size. In practice, Tapestry provides a delivery time within a small factor of the optimal delivery time [24]. Previous work has leverage Tapestry routing for application-level multicast [25] and suggested performance enhancements for wide-area operation [23].

Each Tapestry node or machine can take on the roles of *server* (where objects are stored), *router* (which forward messages), and *client* (origins of requests). We assume that Tapestry nodes, especially routers and servers, are well-connected over high bandwidth links. Nodes in Tapestry have names, Globally Unique IDentifiers (GUIDs), independent of their location and semantic properties, in the form of random fixed-length bit-sequences represented by a common base (*e.g.*, 40 Hex digits representing 160 bits). The system assumes entries are roughly evenly distributed in the node ID namespace, which can be achieved by using the output of secure one-way hashing algorithms, such as SHA-1 [16].

3.1 Prefix-based Routing

Tapestry uses local routing maps at each node, called *neighbor maps*, to incrementally route overlay messages to the destination ID digit by digit (*e.g.*, `8***` $\Longrightarrow$ `89**` $\Longrightarrow$ `895*` $\Longrightarrow$ `8954` where *'s represent wildcards). This approach is similar to longest prefix routing in the CIDR IP address allocation architecture [14]. A node N has a neighbor map with multiple levels, where each level represents a matching prefix up to a digit position in the ID. A given level of the neighbor map contains a number of entries equal to the base of the ID, where the i^{th} entry in the j^{th} level is the ID and location of the closest node which begins with prefix(N, $j-1$)+"i". For example, the 9th entry of the 4th level for node `325AE` is the node closest to `325AE` in network distance that begins with`3259`.

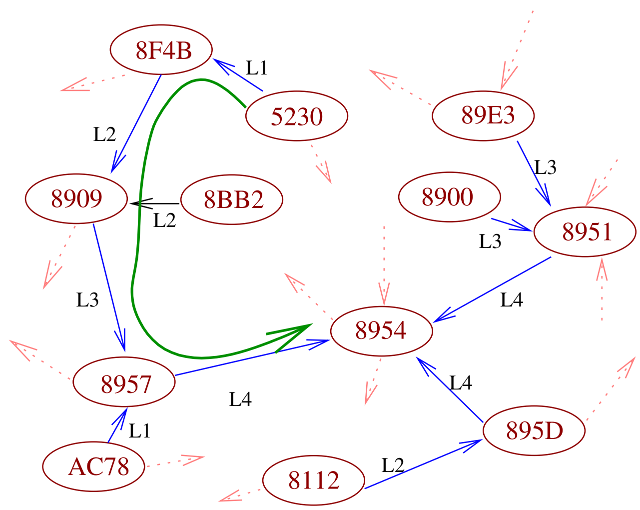


Figure 1: *Tapestry routing example.* Here we see the path taken by a message originating from node `5230` destined for node `8954` in a Tapestry network using 4 hexadecimal digit names (65536 nodes in namespace).

When routing, the n^{th} hop shares a prefix of at least length n with the destination ID. To find the next router, we look at its $(n+1)^{\text{th}}$ level map, and look up the entry matching the value of the next digit in the destination ID. Assuming consistent neighbor maps, this routing method guarantees that any existing unique node in the system will be found within at most $Log_b N$ logical hops, in a system with N nodes using IDs of base b. Because every single neighbor map at a node assumes that the preceding digits all match the current node's prefix, it only needs to keep a small constant size, b, entries at each route level, yielding a neighbor map of fixed constant size $b \cdot Log_b N$.

A way to visualize this routing mechanism is that every destination node is the *root node* of its own tree, which is a unique spanning tree across all nodes. Any leaf can traverse a number of intermediate nodes en route to the root node. In short, the hashed-suffix mesh of neighbor maps is a large set of embedded trees in the network, one rooted at every node. Figure 1 shows an example of hashed-suffix routing. This hierarchical view of Tapestry routing is key to making our fault-tolerant mechanisms efficient, and will be discussed in more detail in Section 4.

3.2 Node Insertion and Deletion

The basic Tapestry infrastructure includes mechanisms to handle changes in the set of nodes that participate in the overlay. For instance, when new nodes join the network, they initiate an integration algorithm that builds neighbor links and informs the rest of the network that they exist. This algorithm is described in detail elsewhere [5]. Essentially, new nodes start by contacting established nodes, then proceed by using the routing mechanism to explore the Tapestry routing mesh. It is during the integration process that existing nodes are given the chance to select the new node as a potential router.

Well behaved nodes have the opportunity to perform a *voluntary deletion* operation by informing the rest of the network before exiting. Alternatively, nodes that cease to behave well are simply removed as potential routes by upstream nodes – using some of the same adaptive mechanisms described in the next section for faulty links.

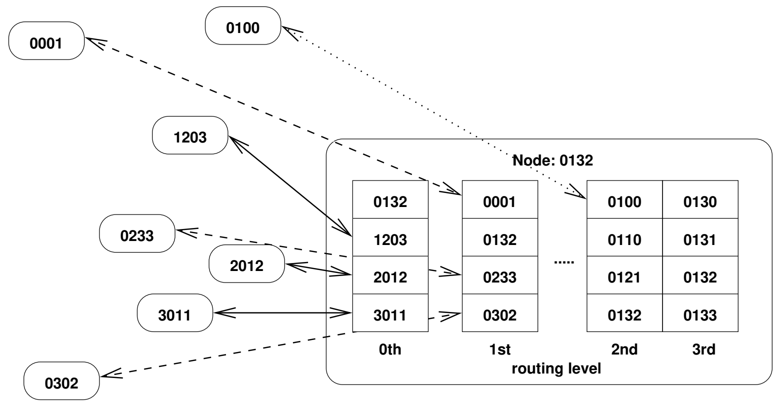


Figure 2: *Keep-alive UDP Beacons.* A diagram of UDP beacons sent by node `0132` in a Tapestry network using four digits of base four. Node `0132` sends periodic beacons to the nodes in its routing table, and each receiver is responsible for sending back periodic aggregate acknowledgments.

3.3 Redundancy Primitives

In addition to providing a scalable routing mechanism, Tapestry also provides a set of fault-tolerance primitives that allow Tapestry routers to quickly detect and adapt to link and node failures.

Fault Detection To adapt to faults in a timely basis, routers monitor links and nodes for failures. Tapestry provides timely link and and node failure detection by using a soft-state model to maintain valid pointers that make up the routing mesh. Routers implement soft-state using periodic broadcasts of information with limited lifetimes of validity. More specifically, Tapestry uses UDP-based beacon messages sent at regular intervals to probe the condition of overlay network links (*i.e.*, reachability, delay, and loss) and return an estimate of current condition of each hop link.

Figure 2 shows the soft-state beaconing mesh. A node X sends periodic UDP probe/beacon messages every T_{probe} seconds to each node Y in its routing table, and each node Y sends back an acknowledgment packet reporting the number of probes received and lost in the previous measurement window. When a new node N inserts itself into the Tapestry, it incrementally builds up a routing table, notifying each entry as it constructs the table. The existing nodes use this notification message to optimize their own routing tables, and add N to their "backpointer" lists.

Each node Y that receives UDP beacon packets from node X uses sequence numbers in the beacons to detect dropped beacons. It keeps a small bitmap representation of a FIFO queue marking the last n packets received (*e.g.*, 16 bits for the last 16 beacons) and the timestamp contained in the last beacon that was received. At regular intervals that are an integer multiple of the probe period, node Y sends the bitmap and timestamp back to probing node X. Node X then uses the bitmap and the timestamp to generate an estimate of current link reachability and quality. Note that the mechanism would be more efficient and responsive if node Y sent UDP probe packets to node X. However, because of the asymmetry of network routing, UDP probe packets must be sent in the same direction as normal unidirectional Tapestry links and messages.

The beaconing and acknowledgment periods are parameters of each particular network, and they are dynamically and introspectively adjusted to minimize bandwidth utilization, while providing reasonably rapid fault detection. For example, in a global Tapestry network using node IDs of 160 bits or 40 hexadecimal digits, we would expect a maximum of 2^{80} nodes before name collision becomes an issue, according to the birthday paradox. With a random distribution of node IDs in this namespace, each node would have roughly

20 levels in their routing table each filled with a maximum of 15 unique entries. If each node sends 100-byte beacon packets every 6 seconds to each its neighbors, the total traffic generated by each node is:

$$BW_{\text{mon}} = 20 \text{ levels} * 15 \text{ entry/level} * 10 \text{ Msg/min} * 120 \text{ bytes/Msg} = 352 \text{ KByte/min} = 44 \text{ Kbit/s}$$

For a network with with 2^{80} node (far larger than any realizable network), this is a minor amount of added network traffic for detecting faults. Furthermore, while this amount of added traffic is significant for a client connected via a low-bandwidth uplink (*e.g.*, an Asymmetric Digital Subscriber Link), the traffic is a minor addition for routers and servers, which we expect will be very well connected.

An important factor to examine is the fault detection time, which is a function of the beacon period, the acknowledgment period, and the Round Trip Time, T_{RTT}, between the beacon sender and receiver. Ordinarily, a fault will not be detected until n beacons have been received. Since beacons are sent every T_{probe} seconds, the maximum time to detect a fault will be the sum of entire acknowledgment period, one-half of the round trip time for the first beacon to arrive (or not to arrive), and an additional one-half of the round trip time for the acknowledgment packet to be sent back to the sender. Thus, the maximum fault detection time will be:

$$T_{\text{Detect}} = n * T_{\text{probe}} + T_{\text{RTT}}$$

The fault detection time can be reduced by reducing either n or T_{probe}. Reducing n will result in fewer beacons being received per acknowledgment packet sent, and thus will result in more frequent acknowledgment packets being sent. Likewise, reducing T_{probe} will result in more frequent beacon packets being sent. Introspection could be used to set either n and T_{probe}. During periods of "long" fault-free conditions, the values could be increased, reducing measurement accuracy. Similarly, during "bursts" of packet losses, the values could be decreased to provide more accurate measurement of network conditions.

Redundant Routes While Tapestry uses periodic beacons to provide an estimate of the current link conditions, it also uses explicit redundant routes to exploit the natural redundancy in the underlying network. Tapestry does this by maintaining a small constant number of backup routes for each entry in a routing table. When a router finds that the default route for an outgoing message is unacceptably lossy, the router switches the message to one of the backup routes. These backup routes are filled in using the same insertion process as the default routes during the node insertion process. The backup routes are the next nearest nodes in terms of network latency that satisfy the prefix-routing constraint.

When the primary link becomes available and reliable, the node switches back to the primary link. Note that to reduce the likelihood of routing flaps, the switching mechanism includes some random hysteresis in switching back to the primary link. The hysteresis is provided by having the node wait for a random number (in the range of 2 to 4) of above-threshold acknowledgments to be received before declaring a link to be reliable and available for routing. By including a random delay, Tapestry reduces the likelihood of a "thundering herd" effect of several nodes switching to a link and rendering a link unusable.

Note, however, that the use of backup routes is not without a cost. Each additional backup route adds to the storage required for the routing table, and it also requires additional bandwidth for beacon probe packets. Therefore, we should balance the number of backup routes per routing entry necessary to maximize reachability against the storage and bandwidth overhead. In previous work [25], we found that maintaining two backup routes per routing entry provides near perfect reachability under link failures.

Thus, assuming each node keeps two backup routes per entry, the bandwidth utilized for UDP beacon probes on our global network would be:

$$BW_{\text{3mon}} = BW_{\text{mon}} * 3 \text{ Msg/entry} = 3 * 352 \text{ KByte/min} = 1055 \text{ KByte/min} = 132 \text{ Kbit/s}$$

This estimate of probe bandwidth is high enough to become prohibitive for nodes with less available bandwidth. We propose two simple optimizations: message piggybacking and reduced probing on backup links. The first optimization is to piggyback beacons onto normal Tapestry messages. For outgoing routes that are sufficiently frequently used by normal message traffic, no additional UDP probes are necessary. Therefore, only less frequently used routes, such as backup routes, require regular probes. Furthermore, for backup links, we can dynamically reduce the the probe rate, trading reduced monitoring accuracy for reduced bandwidth. If beacon loss occurs, the monitoring rate can be increased to the full rate for increased accuracy.

It is also again important to note that this scenario assumes a deployed network of 2^{80} nodes. We expect real deployments of Tapestry to produce significantly less probe traffic. For a network of 2^{32} or roughly two billion nodes (to avoid name collision), the added traffic rate per node is only 26.4 Kbit/s. This rate is sufficiently low for all but the slowest links.

4 Fault-tolerant Routing Mechanisms

In this section, we provide a detailed description of the network structure and algorithms that provide fault-resilient packet delivery in a Tapestry network. There are two key mechanisms: dynamic route selection using precomputed backups, and constrained ("short-distance") multicast and convergence. We describe different types of network failure scenarios, followed by a discussion of each of the fault-tolerant mechanisms and how each one increases the likelihood of successful packet delivery. Finally, we discuss a third application-level mechanism, node-based GUID aliasing, that provides redundancy using an higher-level, orthogonal approach that is independent of the network and the routing namespace.

As outlined previously, we consider two main types of failures that result in loss of availability. First, we consider the scenario where a single network link has failed between directly connected nodes A and B. The impact of this type of failure is limited to flows that cross that single link. In the large majority of these cases, packets can be successfully delivered via a secondary route to B. A second instance is when a network router fails to deliver packets to the next hop on the packet's route, either due to a hardware failure or software misconfiguration. Router failures impact all incoming flows to a node. Successful packet delivery may require routing further away from the failure, with later convergence back to the original route past the failed router and affected nodes.

4.1 Destination-rooted Hierarchies and Convergence

Before we delve into the details of fault-tolerant Tapestry routing, it helps to understand the intuition behind Tapestry routing. Tapestry routing is inspired by previous work on object location [12] and is similar to a modified form of hypercube routing. An alternative way to view Tapestry routing is from the perspective of the destination node, by viewing the routing mesh as a union of *destination-rooted hierarchies*.

As described in Section 3, the basic routing mesh constructed as part of a Tapestry network of N nodes can be seen as a union of N routing trees. From the perspective of a single node X, node X is the root of a spanning tree connecting all nodes in the network. Thus, every traversal from some node Y up the tree to the root is the path taken by Y to X.

Figure 3 shows an example of a sparsely populated network as seen from the perspective of the destination node `0213`. The interesting implication here is that when using node IDs of base b, traffic to and from a single node is constrained to go through a finite set S of nodes, where $|S| = b^h$.

The intuition here is as follows: since the set size at a given level is a function of the distance in hops, h, from the node, the number of possible nodes for its next hop decreases by a factor of b, as a message routes towards its destination. This reduction is due to the requirement that the next hop node must match an

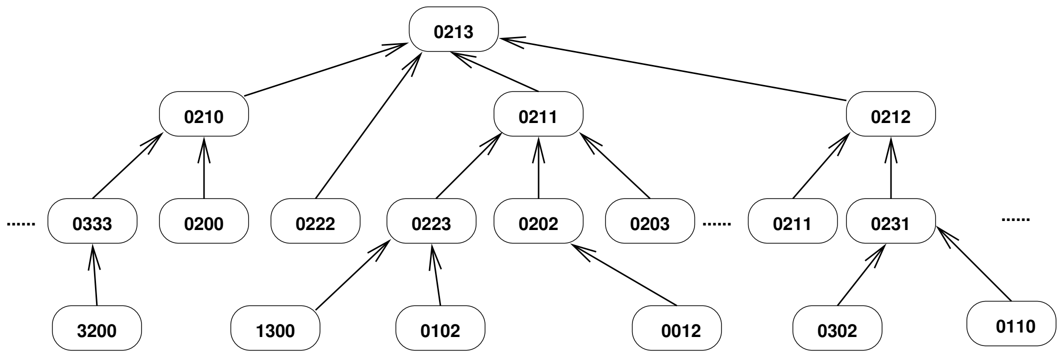


Figure 3: *Example of a destination rooted spanning tree.* An example of a sparsely populated network using node IDs of 4 digits of base 4. Each node points to the closest node (in network latency) that shares an additional digit of a common prefix with the destination node `0213`.

additional prefix digit of the destination node. As the distance to the destination decreases, there is a smaller number of nodes providing the next hop; and thus, messages from nearby nodes are likely to *converge* to or intersect with the same next hop router.

While Figure 3 shows one perspective of the basic Tapestry routing mesh, a interesting result becomes apparent when we examine the impact of convergence on backup routes. By definition, backup routes must point to the next closest (latency-wise) nodes whose node IDs match one more digit to the destination node in its prefix. Thus, it follows that routers pointed to by backup routes will be reasonably close (again, latency-wise) to routers pointed to by primary routes. This observation combined with the convergent property of the Tapestry mesh that we discussed in the previous paragraph, leads us to expect that traffic which is diverted to a backup node will rapidly converge with the original traffic path (most likely on the next overlay hop).

Figure 4 demonstrates this property geometrically. Nodes are laid out in figure to correlate geometric distance between nodes with network latency. As a message travels "up the tree" towards the destination, there are fewer routers satisfying the routing constraint. Simultaneously, Tapestry's locality-based routing property means that the inter-node latency between routers higher in the tree will dramatically increase. In combination, the reduction in satisfying routers and locality-based routing leads to the convergent property shown in the figure. The figure also shows nodes maintaining the primary route and two backup routes, sorted in order of network latency. The routing path from node `1111` to `2222` is highlighted for clarity.

4.2 Route Selection

In Section 3.3, we described how Tapestry detects faults with UDP probes and precomputes backup routes for each entry in the routing table. We now examine the issue of combining link fault-detection with redundant routes to provide a high probability of successful message delivery with minimum added communication overhead. Ideally, routing should use default routes (*i.e.*, those with lowest next-hop latency) whenever possible, since backup routes could possibly lead to longer end to end latencies[2].

While we could design an arbitrarily complex protocol for routing messages under lossy conditions, we start instead with a simple simple protocol, that we call *First Reachable Link Selection* (FRLS), that uses a rough granularity to categorize failures. FRLS defines a global threshold constant value and links with UDP probe results that show a delivery rate lower than this threshold value are marked as `DOWN`, while all other links are marked `UP`. Note that hysteresis is applied in the form of a random interval of valid values that must

[2] In some situations, using a backup route near the source yields an end to end latency that is less than the primary route.

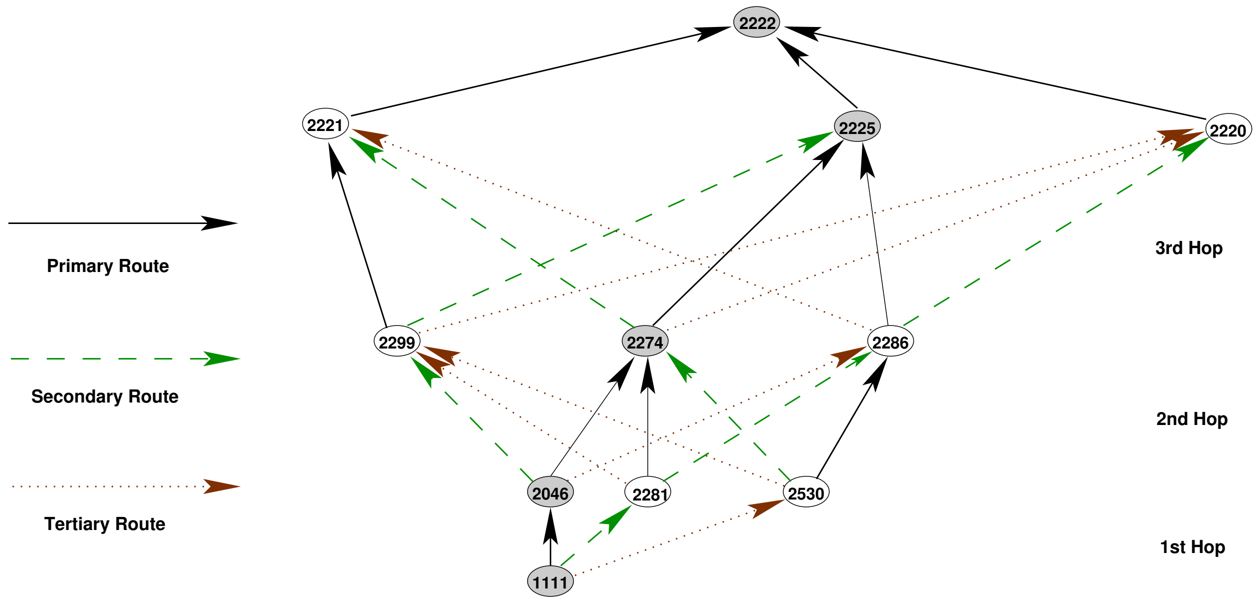


Figure 4: *Routing Hierarchies in Tapestry.* A partial snapshot of a routing mesh from node `1111` to `2222` with backup routes included. Routes are marked with respect to the originating node. Note that with each hop, the expected number of available routers decreases and they are more sparsely distributed.

be seen in order for a `DOWN` link to be marked `UP` again. When a message is ready to be routed, the router examines the default and then backup paths in order of smallest latency first, and forwards the message out on the first `UP` route.

In Figure 5, we see the results of running FRLS when one or two routes at a single route are marked `DOWN`. The figure shows how the resulting path quickly converges with the original routing path after circumventing the failed links. Any additional failures are handled in a similar fashion. An important evaluation metric for a fault-tolerance mechanism is how quickly the new path converges with the original path. For FRLS, the convergence distance is a function of the number of link failures. We examine the convergence behavior using FRLS via simulation in Section 5.

Note that the general problem of route selection is common to all protocols that use routing redundancy to route around failures, including BGP [15]. In particular, the same fault resilience approach can be applied to other self-organizing, scalable overlay network [9, 13, 17, 21]. It is also important to observe that while FRLS and protocols similar to it are useful for routing around congestion [3], the simple UDP measurement scheme was designed to most effectively deal with link connectivity loss, and not congestion measurement.

FRLS focuses on simplicity and low computational overhead, however, we are evaluating algorithms that compare link conditions with finer granularity and make decisions using more complex routing policies. We believe that this clearly is an area in need of further exploration.

4.3 Multicast and Convergence

While FRLS is a more general algorithm for utilizing redundant routing paths, we now discuss a routing algorithm that exploits Tapestry's rapid convergence property. Instead of applying a policy that chooses an alternative route for a message, we propose the notion of *constrained multicast*, a protocol that, when it encounters a faulty link, actively duplicates a message, sends the message copies down multiple paths, and

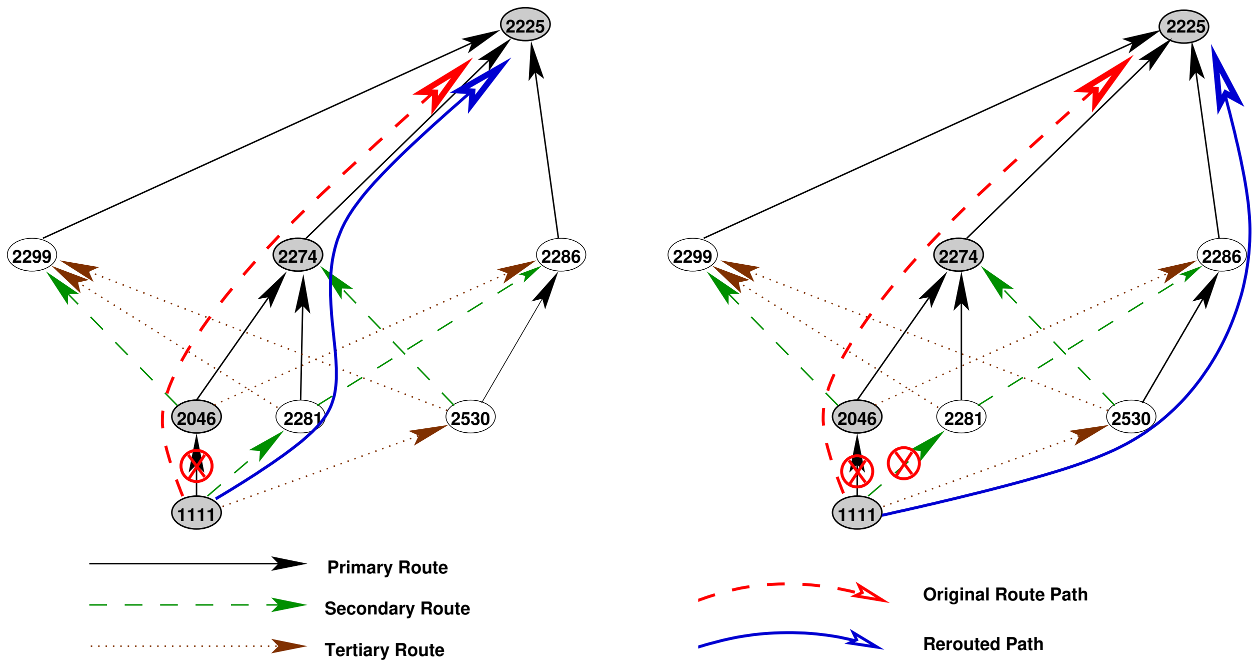


Figure 5: *Routing Around Link Failures with FRLS.* An example showing a portion of the route path from node `1111` to `2222`. When link delivery rate falls below the global threshold constant rate, FRLS defaults to the next available link. A single link "failure" results in a route-around path that converges relatively quickly to the original path (left); while the use of the tertiary link (2^{nd} backup) results in longer time to convergence with the original path.

then utilizes Tapestry's rapid path convergence to enable the duplicates to be dropped on the other side of the fault.

More specifically, when a message arrives at a Tapestry router using constrained multicast, the router examines the link conditions for its primary route. If the link is marked DOWN, then instead of choosing a single outgoing route from a backup route that is marked UP, the router duplicates the message N times and sends the copies across those backup routes that are marked UP, in order of increasing hop latency.

We have already described the intuition behind the convergence property in Tapestry routing. Given this property, we assert that a message sent out on a backup route will quickly (within 1 to 2 additional hops) converge back to the original path to the destination (see Section 5 for results that confirm this assertion).

As the duplicate messages converge, they are identified by the unique sequence number of the original message and any duplicates are dropped. Because of the rapid convergence property, each node only needs to maintain a small list of expected sequence numbers of Tapestry messages, which the node can then use to determine whether a newly arrived message is a duplicate of a previously received message. This list can cover a large time window while minimizing storage overhead by using an efficient index to store sequence numbers (*e.g.*, a starting sequence number and a bitmap of received messages).

Using constrained multicast, a packet can be actively duplicated before crossing a semi-reliable link which significantly increases the probability of successful delivery, while also reducing latency and variance in message inter-arrival time. The price for this benefit is the additional bandwidth is used by duplicate messages, but, rapid convergence tends to both minimize and localize extra bandwidth usage. Figure 6 shows two examples of constrained multicast occurring at different points in the routing path.

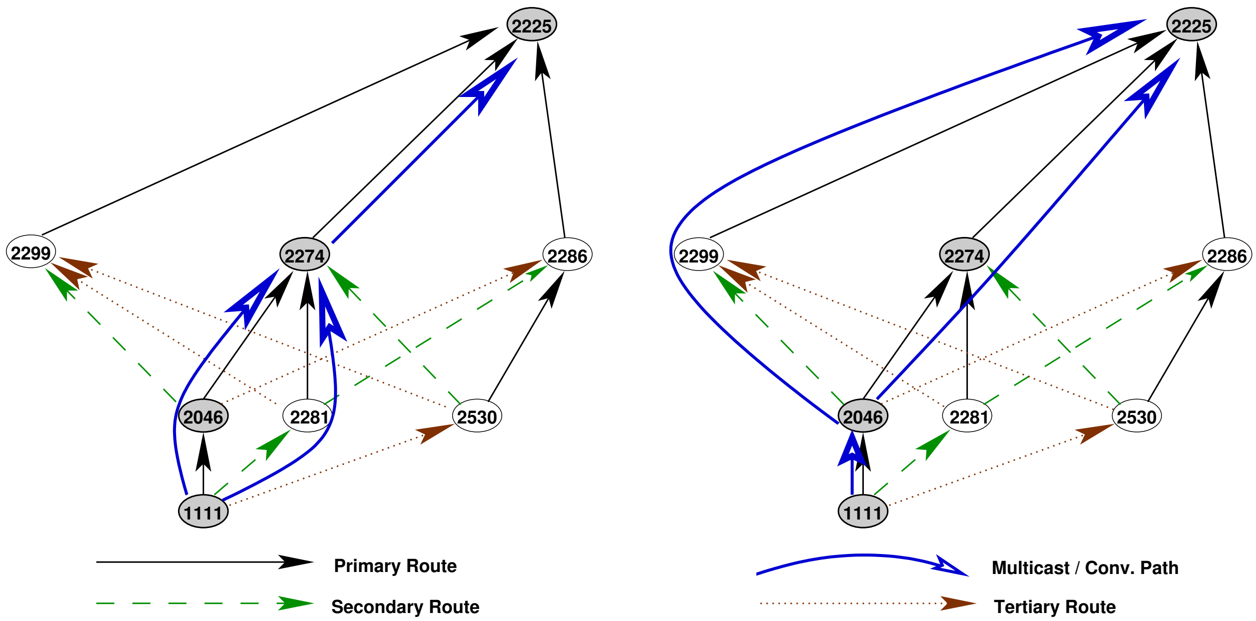


Figure 6: *Constrained Multicast and convergence of messages around link failures.* On the left, multicast occurs at node `1111` and two copies of the message are sent to `2046` and `2281`. On the right, the branch position of the multicast occurs at the next hop at `2046`. Note that a multicast occurring later on the path is expected to converge slower and incur a higher penalty in bandwidth and latency.

This fault-resilience mechanism lets routers use more complex routing algorithms. For instance, a message with a choice between three outgoing lossy routes can use a probabilistic formula to determine which routes to send a copy through to maintain a constant "Expected Copies to Arrival (ECA)" target value. This measure could be used as an adjustable knob to provide different levels of reliability for different traffic types. For example, consider two messages going out on the same route entry. One is a streaming audio packet, with a target ECA of 1; the other is part of an email message, with a target ECA of 1.5. For each of the primary and backup routes, let D be fractional link reliability measure from UDP beacons, then the probability P of sending a message out on a given route is constrained as follows:

$$P(R) * D(R) + P(R_{backup}) * D(R_{backup}) + P(R_{backup2}) * D(R_{backup2}) = ECA$$

$$P(R) \propto D(R), P(R_{backup}) \propto D(R_{backup}), P(R_{backup2}) \propto D(R_{backup2})$$

This type of constraint allows routing policies to further specify the relative preference between primary and backup routes, and choose between the performance and overhead tradeoffs.

This type of active duplication is specifically designed to deal with links experiencing intermittent packet drops, such as congestion-related losses; thus, a preliminary objection to the approach should be that this mechanism could further exacerbate congestion by adding more traffic. However, closer inspection should convince the reader that the duplicate packets will traverse an alternate route that circumvents and, thus, should not exacerbate, the current congestion. If the congestion occurs at a physical hop shared by both or all 3 outgoing routes, the same level of loss will be reported on each route, and the probabilistic constraint would then limit outgoing duplicates on those links.

4.4 Node-based GUID Aliasing

The final fault-resilience approach that we discuss is a redundancy approach that we call, *Node-based GUID Aliasing*. This approach is completely orthogonal and independent of the network and the namespace, as it moves the problem up to the application domain by using redundant naming of nodes.

Tapestry's core routing mechanisms are designed to route messages to nodes and node GUID aliasing simply adds an additional layer of indirection. Ordinarily, Tapestry constructs a node's name or GUID by taking the node's IP address or public key and applying a cryptographic hash function (*e.g.*, SHA-1 [16]) to create the GUID. With node GUID aliasing, we use multiple virtual names to refer to the same physical node by creating multiple GUIDs for the node. We do this by hashing the initial value (node IP address or public key) with multiple salt values to create several GUIDs.

The insertion of these alternate GUIDs into Tapestry creates "routing planes" or dissemination trees that are random and, thus, likely to be independent of each other, except at the source and destination nodes. If source and destination nodes are each very well connected (*e.g.*, > 1 network connection) and the network between the two nodes has a more of a mesh nature than a transit-stub one, then the routing planes will be more likely to be completely independent. To use this approach, an application sends duplicate copies of each message to each one of the GUIDs or a subset that it determines by using end to end reliability measurements.

There are two advantages to this approach: it reduces or eliminates the reaction time delay from UDP beacons and acknowledgments and it provides very reliable delivery (with the above connectivity assumptions). However, it is important to note that while this approach provides very reliable delivery, it has a bandwidth cost that is significant — equivalent to the number of duplicate copies.

For networks that are facing congestion instead of reliability losses, this could make the situation worse, not better. Furthermore, the excess bandwidth consumption is not localized as is the case with Tapestry-level multicast. Nevertheless, even though it has a significant cost, this approach is interesting because of its reaction time benefits and because it is easily applicable to other overlay network projects.

5 Measurement and Evaluation

In this section, we present simulation results from measuring the relevant properties of fault-resilient Tapestry routing. After discussing our simulation methodology, we present results showing the efficiency of FRLS at routing around link failures, and the cost of routing around failures as measured in proportional increase in latency. We then examine the overhead or "cost" for taking a backup route (branching). First, we look at how quickly branched paths converge with the previous route path. We then examine the net cost of branches, in terms of additional network bandwidth and additional end-to-end routing latency.

5.1 Simulation Methodology

To simulate Tapestry routing on large-scale ($\approx$ 5,000 nodes) topologies, we implemented a packet-level simulator based on the Stanford Graph Base (SGB) package. We chose transit-stub topologies as the most realistic topology model and, as described below, we used real wide-area measurement data to calibrate the GT-ITM [22] topology generator. The simulator reads in the resulting topology data in SGB format, and stores each node as a Vertex object along with its routing table information. It's important to note that our simulation uses the same Tapestry algorithms as our prototype implementation.

To perform experiments, we built a Tapestry network of 4,096 nodes on the 5,000 node topology. Tapestry nodes were placed randomly in the topology set, and named randomly from a namespace of 6 digit, base 4 (hexadecimal) names. This namespace implies that any point to point Tapestry route will take at most 6

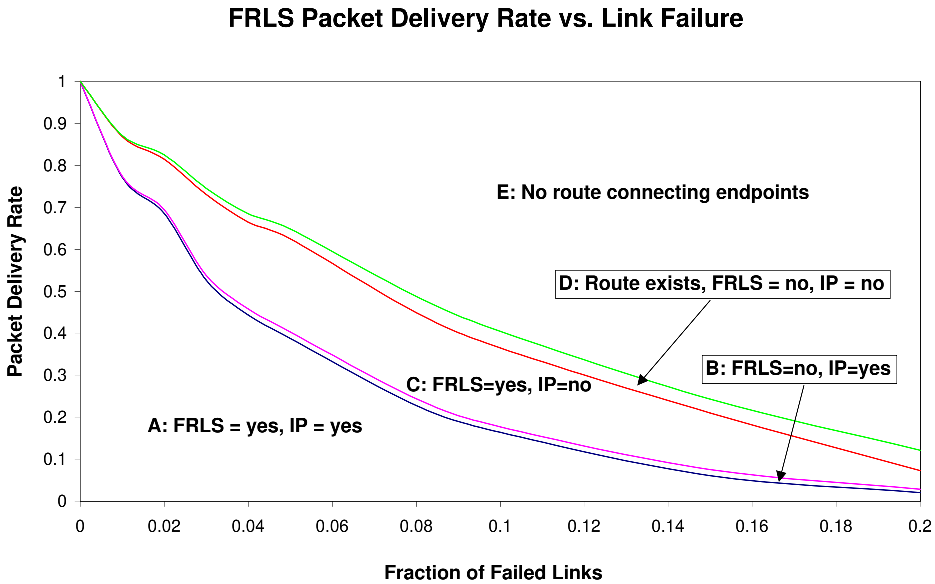


Figure 7: *Reachability of FRLS and IP vs Link Failures.* Simulation results of the probability of successful packet delivery using FRLS and normal IP as underlying link failures increase. We assume link failures result in complete loss, and BGP re-route does not converge quickly enough to sustain connection. This graph shows that FRLS achieves near ideal fault-resilience and a significant improvement over IP.

overlay hops. For each experiment, we generated at least 1,000 values per data point. Finally, we repeated the experiments with multiple choices of overlay node placement to ensure that overlay construction (and Tapestry node placement) did not have an effect on our simulation results.

5.2 NLANR Topology Calibration

To generate results that accurately model large-scale Tapestry behavior on a real wide-area topology, we created a synthetic topology that based upon real measurement results. This was a complex process as, despite substantial large-scale measurement efforts at CAIDA (including the Skitter[3] project), in NLANR's AMP and NAI projects[4], and in numerous academic research projects, obtaining an accurate representative topology of the real Internet with complete latency and connectivity information is currently infeasible.

Instead, we used available data from NLANR to design a representative synthetic topology. We started by extracting topology information from active measurement data from NLANR's AMP project. Their data collection includes topology and Round-Trip Time (RTT) data from 130 active measurement sites. The data consists of 14,269 files, each of which contains a single `traceroute`-based path measurement. From this data, we extracted a network topology with 1,780 unique nodes connected by 3,305 edges. Unfortunately, this dataset was not well suited for running Tapestry experiments, since it effectively provides topology information consisting of single long routing paths organized in a starfish-like formation, where each long path originates from an AMP location. Basically, this topology is missing the interconnectivity information between these long links. This interconnectivity is present in the real Internet (in the form of inter-service

[3]See `http://www.caida.org/tools/measurement/skitter/`
[4]See `http://moat.nlanr.net/infrastructure.html`

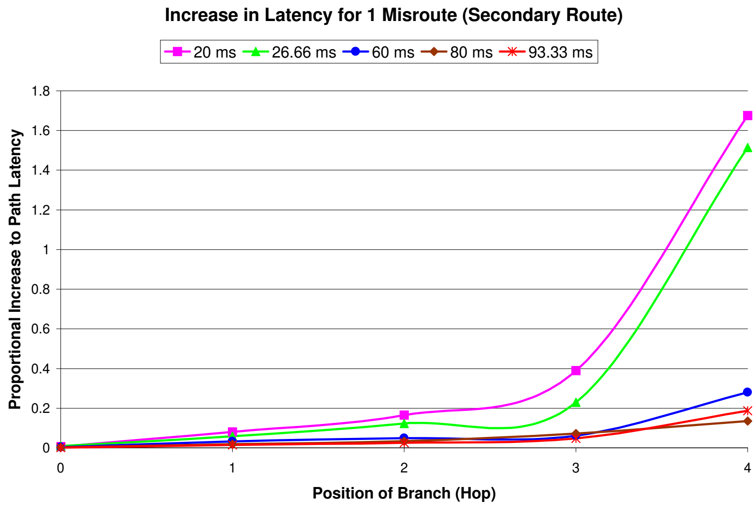


Figure 8: *Latency Overhead of Misrouting (Secondary).*

provider peering and transit relationships) and is crucial for our experiments, since it provides the core mesh-like routing redundancy that makes fault-tolerant Tapestry routing possible.

Since resulting topology could not itself be used for our experiments, we chose the next best alternative, extracting the relevant characteristics from the real measurement data and then using the characteristics to calibrate an artificial topology generator. Our examination of the NLANR AMP data shows that the average latency for local area links is 2 milliseconds and it is 30 milliseconds for wide-area links. Using the AMP data, we also extracted the average latency values for analogous links to a transit-stub topology's transit-transit, stub-stub and transit-stub links. We then used the extracted statistics as inputs to the GT-ITM topology generator to generate a topology of 5,000 nodes for use in our experiments.

5.3 Route Selection

FRLS Delivery Success Rate The first question we explore is: using binary failure mode for all links in a network, does FRLS deliver packets when there exists a reachable path between the endpoints? A binary failure mode means that a link is either delivering packets without loss or dropping all packets (note that we ignore congestion-based losses). For evaluation purposes, we compare basic datagram delivery over IP with basic datagram delivery over IP enhanced by FRLS redirection over Tapestry.

As shown in previous work [7, 10, 11], BGP fault-recovery mechanisms can take tens of minutes to converge to a consistent form yielding significant disruptions in end to end connectivity for applications . For our simulation, we focus on timescales of tens to hundreds of milliseconds and assume that IP cannot re-route around failures on that time scale. As another simplifying assumption, we simulate IP by traversing the shortest path between the endpoints. While this may not accurately represent the reality of BGP policies and hot-potato routing, it results in shorter, and potentially more complete, route paths than real IP, and therefore presents a more failure resilient protocol for comparison.

Using our NLANR-calibrated transit stub topology, we randomly inject link failures into the network, while repeatedly traversing all possible pair-wise communication paths with IP and FRLS. We then categorize the results into one of five categories and plot the portion of all communication paths that falls into each category as a probability graph. The categories are as follows:

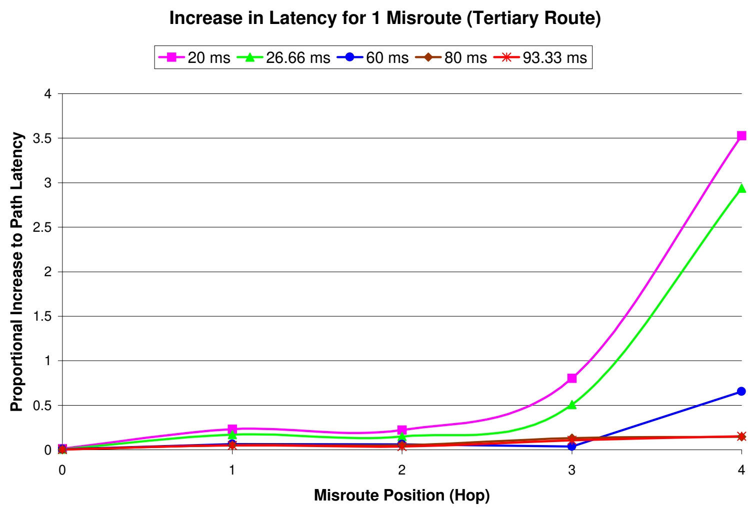


Figure 9: *Latency Overhead of Misrouting (Tertiary).*

1. *A: FRLS=yes, IP=yes*. This is the fault-free case, where the injected faults have no impact on the endpoints chosen. Both IP and FRLS successfully deliver packets.

2. *B: FRLS=no, IP=yes*. This category includes traversals where IP successfully delivers the packet while FRLS fails. This occurrence is relatively unlikely, and occurs because Tapestry overlay routing may travel more hops than IP. If these additional hops include irrecoverable faults (*i.e.*, all exiting routes have failed), then FRLS would fail while IP succeeds.

3. *C: FRLS=yes, IP=no*. This category is interesting because it shows conditions where the IP route is broken due to one or more link failures, and FRLS successfully re-routes around the failure(s).

4. *D: Route exists: FRLS=no, IP=no*. This is the instance where an existing route between the endpoints exists, but both IP and FRLS are blocked by link failures.

5. *E: No route connecting endpoints*. As the fraction of failed links increases in the network, more and more pairs of endpoints become completely partitioned. This category includes all such partitioned pairs.

Figure 7 shows the simulation results. We have performed the same experiment on a variety of network topologies, including randomly generated transit-stub graphs, topologies from TIERS, an autonomous systems connectivity graph, and a graph of MBone nodes, all with similar results. Region C and D represent the failure cases for IP, where a route exists but IP fails to deliver the packet. Our results show that Region C dominates Region D, indicating that FRLS successfully routes around most link failures when working paths exist. This is particularly true when the failure rate is small. Also note that as failures increase, FRLS' delivery success rate is several times that of IP.

Latency Overhead for Misrouting We now take a look at the latency overhead for misrouting around failures. Our intuition is that because misrouting effectively "branches" off from the normal routing path, it would incur a performance penalty in the form of longer end-to-end latency. We would also expect that misrouting through the tertiary route would incur a relatively higher latency penalty. Recall the example illustrated in Figure 5.

If Tapestry routing was ideal (produced the shortest path with RDP = 1), then any misrouting would definitely incur a significant latency penalty. Tapestry overlay routing is non-ideal, however. Determining the next hop is "greedy" in the sense that we choose the closest next hop node with the matching prefix, even though it might actually be further away to the message destination. Therefore, there are times when misrouting actually benefits overall latency, when a locally suboptimal decision leads to a shorter end-to-end path overall.

In Figures 8 and 9, we take 1.2 million randomly chosen unique paths from the topology, and measure the proportional increase in latency experienced when a message misroutes. We plot the result against where the misroute occured (which hop in the overlay route path). Figure 8 shows the results when misrouting via the secondary route, while Figure 9 shows results when misrouting via the tertiary route. As expected, in each case, the proportional increase in latency is higher when misrouting takes place later in the overlay route (at the 3rd or 4th hop). Our results also confirm our intuition that misrouting via the tertiary route generally incurs a larger latency penalty than misrouting via the secondary route.

We note that the proportional increase in latency is much higher for paths of shorter lengths. This is due to the fact that Tapestry routes efficiently in the local area network [24]. Because of the relatively significant jump in latency between LAN links and WAN links, the greedy mesh construction algorithm leads Tapestry nodes to search for nearby routers before venturing outside the stub network or LAN. This results in efficient routes to local destinations, often taking less than the expected number of hops. The message often arrives at the next hop only to find it matches more than the expected number of prefix digits. We refer to these shortcuts as "virtual hops," since the messages need not leave the router to gain additional matching prefix digits of the destination node. In reality, we would never misroute on virtual hops. We do so in this experiment for consistency, resulting in disproportionally large penalties for extremely short paths.

Overall, our results show that misrouting does not penalize the end-to-end latency heavily, resulting in less than a 20% penalty for misrouting on the large majority of secondary routes, and less than 50% for misrouting on tertiary routes.

5.4 Effects of Branching

In this section, we examine the various penalties a message incurs for utilizing backup pointers, in order to better understand the performance and overhead tradeoffs involved in Tapestry fault-resilient routing.

Hops to Convergence. The first metric we examine is the number of hops to convergence. This metric is the number of overlay hops that a message travels across after using a backup route and until it arrives on a router along the original route path. In an ideal Tapestry network on top of a uniform (mesh-style) network with uniform connectivity and distance between neighboring nodes, the expected hops to convergence value will be slightly above 1.

Figure 10 shows our simulation results on our synthetic topology as a function of: the distance between communication endpoints, at which overlay hop on the route the misroute or branch is taken, and which of the secondary or tertiary backup routes is taken. Each line represents misrouting or branching at a secondary or tertiary route at a particular branch position. We see that as expected, hops to convergence values fall between 1 and 2, with tertiary routes taking on average more hops before converging than secondary routes.

Furthermore, misrouting or branching at later hops in the overlay route results in less number of hops before convergence. This confirms intuition, since the number of possible routers decreases with each additional overlay hop, with all hops converging on the final node at the destination.

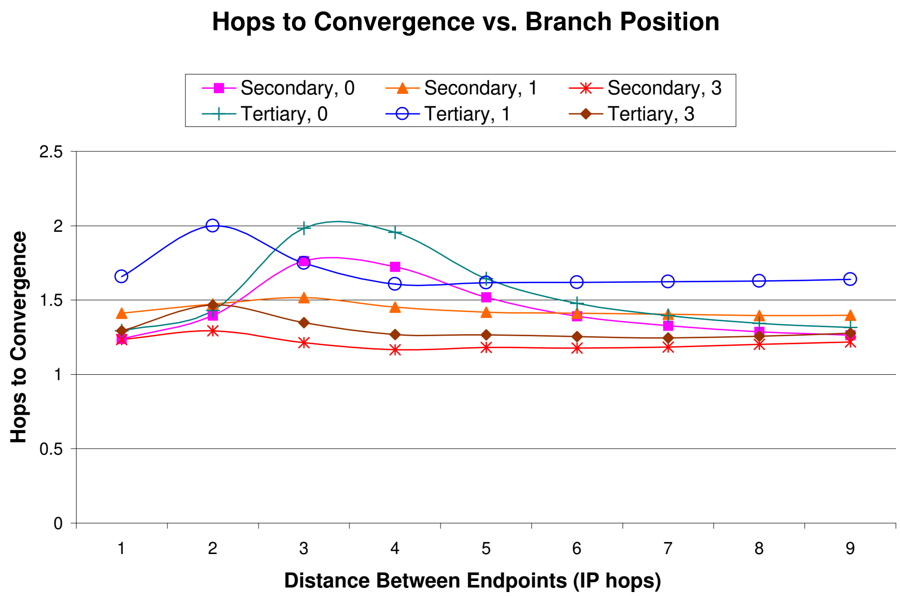


Figure 10: *Hops to Converge vs. Branch Position.* Simulation results showing the average number of overlay hops taken after taking a backup route before converging with the original path. Results are shown for branch positions of 0, 1, and 3 for a Tapestry with 6 hops between nodes. The X axis shows true length (distance in IP hops) between the endpoints.

Bandwidth Cost for Multicast. Next, we examine the tradeoffs with using Tapestry's constrained multicast for greater reliability. In this experiment, we examine 1.2 million randomly selected unique paths in the NLANR-calibrated transit stub topology. This is approximately 7.5% of all possible pair-wise paths. For each of these paths, we simulate the bandwidth used by a single additional multicast packet which is dropped when it converges with the original route path. We repeat this experiment for each hop where we can multicast. On a Tapestry network with 6 digits, each message travels through 5 routers (including the source) before reaching its destination. We then calculate the ratio of the bandwidth overhead to the total bandwidth cost of the original route.

In presenting our results, we categorize all paths according to their actual IP distance in hops. In Figure 11 we see the simulation results for our experiment, where each multicast sent the duplicate packet to the secondary route. As expected, we see that the later in a route that a multicast occurs, the higher the added bandwidth cost. This result occurs because each hop in a Tapestry path is likely to cover more and more IP hops. More importantly, we see that for the most common paths lengths of 8–10 hops, sending a multicast message incurs additional bandwidth of only around 10% of the bandwidth utilized by the normal route.

Also note how the proportional bandwidth cost decreases as the IP distance of the communication points increases. When Tapestry routes between two nearby nodes, it usually takes very few hops, incurring a low RDP. This is the same "virtual hops" phenomenon mentioned in Section 5.3. In reality, a message would not multicast on these virtual hops, and only multicast when leaving the current physical node. To provide uniformity in our experiment, however, we force multicasts at these virtual hops, incurring additional bandwidth that weighs disproportionally large on a short IP path.

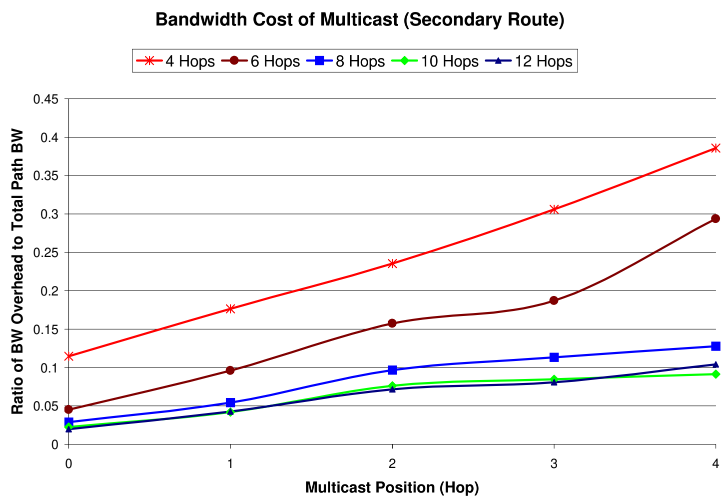


Figure 11: *Bandwidth Overhead of Branching (Secondary).*

6 Discussion

In this paper, we have presented algorithms and simulation results for several techniques that help mitigate the effects of packet loss and congestion hot spots for applications running in the wide-area on the Tapestry network. In this section, we discuss several issues: the broader applicability of these techniques to other DOLR algorithm-based overlay networks; the scalability, stability, and adaptability of our techniques; and the effect on Service Level Agreements and peering agreements of our techniques.

In Section 4, we discuss three algorithms for fault-tolerant routing (FRLS, constrained multicast, and node-based GUID aliasing) in the specific context of Tapestry. We believe that each of these approaches can be applied to varying degrees of success to other DOLR overlay networks, such as Kademlia [9], CAN [13], Pastry [17] and Chord [21]. For example, in Chord, a router can choose between multiple finger pointers for each outgoing hop. Likewise, Pastry uses a similar routing technique as Tapestry, thus a modified form of constrained multicast could be used in a similar fashion. Finally, the use of multiple realities in CAN and the use of multiple node IDs in Chord are equivalent analogies of node-based GUID aliasing.

Two of the three approaches, constrained multicast and node-based GUID aliasing, have the potential to negatively impact the scalability and stability of the underlying Tapestry network. However, our simulation results of constrained multicast show that the scope of its impact on the network is fairly localized — this is exactly where we expect to find abundant bandwidth. GUID aliasing may impose a significant impact on the network, thus, we recommend that it only be used in cases where response time and very reliable delivery are both critical, high bandwidth is available, and a high degree of interconnection is present at the source, destination, and intervening nodes.

In terms of adaptability and responsiveness, GUID aliasing is the best approach, as it continuously sends messages along multiple paths. However, it is a coarse-grained approach. Both FRLS and constrained multicast are limited by the beacon period and the acknowledgment window. Increasing the beaconing period and reducing the acknowledgment window (or supporting polled acknowledgments from the beacon sender) will increase responsiveness, at the cost of higher monitoring traffic.

Finally, an important, but often overlooked, consideration in the choice of alternate routes is the various peering and transit policies of the participating Internet service providers. Overlay networks have traditionally ignored the arrangements, while optimizing the paths that messages take. If overlay networks are to be

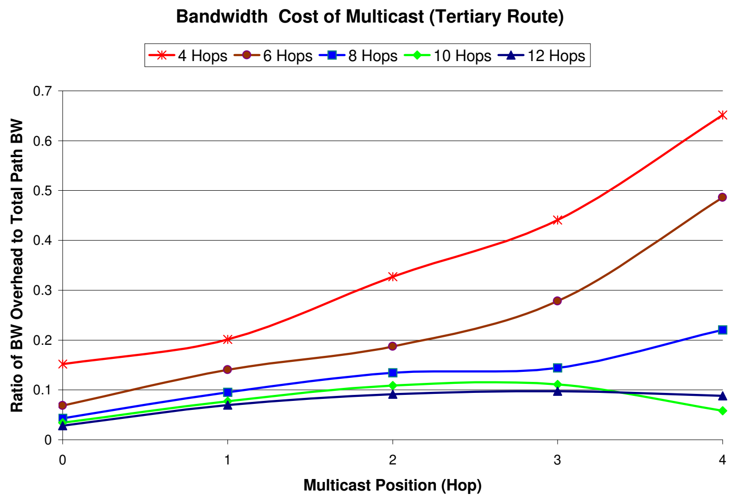


Figure 12: *Bandwidth Overhead of Branching (Tertiary).*

successfully deployed on a large-scale, the routing algorithms will have to take into account the inter-AS routing policies. We expect that this will reduce the alternatives for fault-resilience (and potentially overall connectivity), however, it also presents the opportunity to explore new and interesting short-term peering relationships and Quality of Service brokering relationships.

7 Conclusion

In this paper, we described a technique to provide fault tolerance and high performance through continuous precomputation of alternative pathways and dynamic selection among alternates. We utilized the Tapestry overlay routing network as a framework in which to embody precomputed alternative paths. At each routing hop, the basic Tapestry routing algorithm chooses between optimal or near optimal paths, while soft-state beacons continuously probe the network to pre-compute these alternative paths. We showed via simulation that a simple protocol 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.

The techniques presented in this paper provide an alternative to today's configuration chaos; they provide a framework whereby extra resources – alternative communication paths and router cycles – can be seamlessly exploited to provide stable, high-performance communication. They herald a new age in which designers can capitalize on Moore's law growth to yield a better overall user experience.

References

[1] ANDERSEN, D. G., BALAKRISHNAN, H., KAASHOEK, M. F., AND MORRIS, R. Resilient overlay networks. In *Proceedings of SOSP* (October 2001).

[2] CHANDRA, B., DAHLIN, M., GAO, L., AND NAYATE, A. End-to-end WAN service availability. In *Proceedings of USITS* (March 2001), USENIX.

[3] CHESHIRE, S., AND BAKER, M. A wireless network in mosquitonet. *IEEE Micro 16*, 1 (February 1996), 44–52.

[4] DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with CFS. In *Proceedings of SOSP* (October 2001).

[5] HILDRUM, K., KUBIATOWICZ, J. D., RAO, S., AND ZHAO, B. Y. Distributed object location in a dynamic network. In *Proceedings of SPAA* (Winnipeg, Canada, August 2002), ACM.

[6] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS* (November 2000).

[7] LABOVITZ, C., AHUJA, A., ABOSE, A., AND JAHANIAN, F. An experimental study of delayed internet routing convergence. In *Proceedings of SIGCOMM* (August 2000).

[8] LABOVITZ, C., MALAN, G. R., AND JAHANIAN, F. Internet routing instability. *IEEE/ACM Transactions on Networking 6*, 5 (1998), 515–526.

[9] MAYMOUNKOV, P., AND MAZIERES, D. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of IPTPS* (March 2002).

[10] PAXSON, V. End-to-end routing behavior in the internet. In *Proceedings of SIGCOMM* (Stanford, CA, Aug. 1996), ACM, pp. 25–38.

[11] PAXSON, V. End-to-end internet packet dynamics. In *Proceedings of SIGCOMM* (Cannes, France, Sept. 1997), ACM, pp. 139–152.

[12] PLAXTON, C. G., RAJARAMAN, R., AND RICHA, A. W. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of SPAA* (June 1997), ACM.

[13] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. A scalable content-addressable network. In *Proceedings of SIGCOMM* (August 2001).

[14] REKHTER, Y., AND LI, T. An architecture for IP address allocation with CIDR. RFC 1518, `http://www.isi.edu/in-notes/rfc1518.txt`, 1993.

[15] REKHTER, Y., AND LI, T. A border gateway protocol 4 (BGP-4). *IEEE Micro 19*, 1 (Jan. 1999), 50–59. Also Internet Engineering Task Force, RFC 1771.

[16] ROBSHAW, M. J. B. MD2, MD4, MD5, SHA and other hash functions. Tech. Rep. TR-101, RSA Laboratories, 1995. version 4.0.

[17] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware* (November 2001), ACM.

[18] ROWSTRON, A., AND DRUSCHEL, P. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of SOSP* (October 2001).

[19] ROWSTRON, A., KERMARREC, A.-M., DRUSCHEL, P., AND CASTRO, M. SCRIBE: The design of a large-scale event notification infrastructure. In *Proceedings of NGC* (November 2001).

[20] SAVAGE, S., ET AL. Detour, a case for informed internet routing and transport. *IEEE Micro 19*, 1 (January 1999), 50–59.

[21] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM* (August 2001).

[22] ZEGURA, E. W., CALVERT, K., AND BHATTACHARJEE, S. How to model an internetwork. In *Proceedings of IEEE INFOCOM* (1996).

[23] ZHAO, B. Y., DUAN, Y., HUANG, L., JOSEPH, A., AND KUBIATOWICZ, J. Brocade: Landmark routing on overlay networks. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (March 2002).

[24] ZHAO, B. Y., KUBIATOWICZ, J. D., AND JOSEPH, A. D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Division, April 2001.

[25] ZHUANG, S. Q., ZHAO, B. Y., JOSEPH, A. D., KATZ, R. H., AND KUBIATOWICZ, J. D. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of NOSSDAV* (June 2001).